Point Cloud Analysis for ML-Based Malicious Traffic Detection: Reducing Majorities of False Positive Alarms

Chuanpu Fu
Tsinghua University
Beijing, China

Qi Li
Tsinghua University
Beijing, China

Ke Xu
Tsinghua University
Beijing, China

Jianping Wu
Tsinghua University
Beijing, China

ABSTRACT

As an emerging security paradigm, machine learning (ML) based malicious traffic detection is an essential part of automatic defense against network attacks. Powered by dedicated traffic features, the ML based methods can detect various sophisticated attacks, in particular capturing zero-day attacks, which cannot be achieved by the traditional non-ML methods. However, false positive alarms raised by these advanced ML methods become the major obstacle to real-world deployment. These methods require experts to manually analyze false positives, which incurs significant labor costs. Thus, it is vital that we can reduce such false positives without heavyweight manual investigations.

In this paper, we propose pVoxel, an unsupervised method that identifies false positives for existing ML based traffic detection systems without requiring any prior knowledge on the alarms. To effectively process each alarm, pVoxel treats the traffic feature vector associated with the alarm as a point in the traffic feature space, and utilizes point cloud analysis to capture the topological features among the points for classifying the alarms. In particular, we aggregate the points into voxels, i.e., high-dimensional cubes, which allows us to develop an unsupervised method to identify the voxels indicating false positives according to their density features. Our experiments with 75 real-world datasets demonstrate that pVoxel can effectively reduce 95.55% false positives for 11 state-of-the-art traffic detection methods under various settings. Meanwhile, pVoxel can handle 201.10 thousand alarms per second, which demonstrates that it can achieve efficient alarm processing.

CCS CONCEPTS

• **Security and privacy** → **Intrusion detection systems**.

KEYWORDS

Machine learning; malicious traffic detection; point cloud analysis

1 INTRODUCTION

Machine learning (ML) based malicious traffic detection is an emerging security paradigm, which identifies attack traffic by learning the features of traffic [12]. It is essential for detecting various sophisticated attacks [8, 32, 39, 75, 80, 97] and compensates for the unknown attack (i.e., zero-day attacks [31, 59]) detection capabilities of the traditional rule based detection [78, 89, 92, 95].

However, false positive (FP) alarms [1], i.e., the alarms triggered by benign traffic, significantly plague these ML based detection systems. For example, in a network with terabit-scale traffic [31, 32, 97], millions of benign flows may trigger a huge number of FPs [4]. On the other hand, when cooperating with defense systems [54, 88, 89, 95] to filter out attack traffic, FPs, as high-cost errors, incur collateral impacts on benign traffic [4, 76]. Moreover, existing studies show that FP alarms may reach 99% [1], which has hindered the real-world deployment of ML based detection [48]. Thus, it is vital for ML based detection systems to effectively handle such false positives [76].

The existing studies reduce FPs by applying retraining [24, 25] or whitelists [11, 80]. Table 1 summarizes the drawbacks of these methods. Overall, these methods require a huge effort from human experts to identify majorities of FPs. For example, the retraining methods require manually identifying FPs and including them in the training set to retrain models [24, 25], while the whitelist methods craft fixed rules to exclude similar FPs according to manually identified FPs [80]. However, it is almost impossible to manually identify the FPs [1, 4, 76] due to the high number of alarms [1, 83]. In addition, these methods merely reduce the FP alarms that are similar to the manually identified FP alarms but cannot handle unforeseen FP alarms that cannot be identified by experts.

In this paper, we set out to automatically identify FPs among the massive alarms without requiring human investigations. We capture these FPs by identifying the distinct distribution of the corresponding malicious traffic features in the traffic feature space. Specifically, benign traffic features associated with FPs are generated by diverse benign user behaviors, which are sparsely distributed in the feature space [28]. In contrast, flows generated by attack tools are similar and have densely distributed traffic features, which are associated with true positives (TPs) [32]. Thus, we can utilize the topological features that capture the distinct densities to differentiate FPs and TPs.

We develop pVoxel, a system that aims to automatically identify FPs for ML based traffic detection systems. To achieve this, pVoxel treats the feature vector associated with each alarm as a point in the traffic feature space. Particularly, it leverages point cloud analysis [67] to capture the topological features among the points, and learns the features in an unsupervised manner, which allows pVoxel to identify FPs without any prior knowledge (e.g., manually

Chuanpu Fu, Qi Li, Ke Xu, and Jianping Wu

Table 1: Comparison with existing FP reduction methods.

Properties \ Methods		Retraining	Whitelist	pVoxel
Required Resources	Human Experts	Required	Required	Not Required
	Training Datasets	Required	Not Req.	
	Testing Datasets	Not Req.	Required	
	Benign IP Lists	Not Req.	Required	
	ML Models	Required	Not Req.	
Known Issues	Forgetting [25]	With	Without	Without
	Evasion [68]	Without	With	
Runtime Performance	Low Labor Costs	✗	✗	✓
	High Efficiency	✗	✗	✓
	Low Latency	✗	✗	✓
	Unforeseen FPs	✗	✗	✓

identified FPs [4]). By this way, pVoxel does not interfere with model training and thus avoids the catastrophic forgetting issue incurred by retraining [25], i.e., models forget how to detect attacks after retraining. Moreover, pVoxel does not rely on the testing set information (e.g., benign IP addresses [80]) that is required by the traditional whitelist methods, because they reduce FPs according to IP addresses instead of traffic features. Besides, point cloud analysis can efficiently extract the topological features [67], which allows pVoxel to handle a large number of alarms with low latency.

To efficiently perform the point cloud analysis, we extend traditional voxel analysis in the 3D space [34, 53] and design a three-step approach that achieves voxel analysis in the high-dimensional traffic feature space. First, we normalize the feature vectors to construct points and use a voxel to represent the points within a cube in the high-dimensional feature space. Meanwhile, we identify isolated points that denote the FPs triggered by infrequent benign traffic patterns. Notably, a voxel can represent many points and thus significantly reduces the processing overhead. Second, we utilize a *community* to represent a group of adjacent voxels, which further reduces the overhead. Third, we extract density features for each community and utilize unsupervised learning to detect the communities with significantly high densities that denote TPs triggered by massive attack flows with similar traffic features. The alarms denoted by the rest of the communities are labeled as FPs.

Furthermore, we develop a *stochastic geometry* model [21, 77] to prove the effectiveness of pVoxel. It models the location of the points in the traffic feature space and aims to obtain the expected density of the points represented by a voxel. We theoretically prove that the features of traffic generated by attack tools exhibit dense distributions, which will be represented by the voxels with high densities. We show that the features of the traffic generated by diverse benign user behaviors illustrate sparse distributions, which are captured by the voxels denoting FPs with low densities. Therefore, pVoxel can effectively classify the alarms into TPs and FPs according to the density features of the voxels. We provide several theoretical bounds on the density for five typical traffic features [8, 11, 31, 75, 97], which lay the foundation of efficient FP reduction.

We prototype pVoxel with NVIDIA's CUDA parallel computing platform for GPUs [63]. To extensively evaluate the performance of pVoxel, we prototype 11 state-of-the-art traffic detection systems including flow based [8, 75], packet based [39, 59], frequency based [31, 33], and graph based methods [32]. We replay 75 traffic datasets collected from eight different networks on a real-world

(a) False Positive Rate (FPR). (b) Number of FPs.
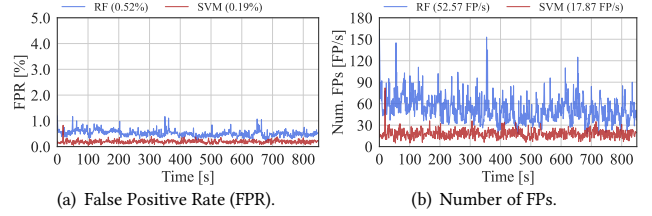
Figure 1: Benchmark FPR and the number of FPs.

testbed [17, 18, 31–33, 59, 97] and obtain real FP alarms. The experiment results illustrate that pVoxel can reduce 95.55% FPs on average without any prior knowledge. Meanwhile, the side-effect of TPR decrease is only 2.62%. Moreover, pVoxel is able to increase 14.67% AUC, and significantly improves the performance with regard to seven other widely used accuracy metrics. Our experiments also demonstrate that pVoxel reduces 5.05 times FPs over the traditional retraining methods. In particular, pVoxel is robust to various ML models with many hyper-parameter settings. Furthermore, pVoxel achieves over 201.10 thousand alarms per second processing throughput with an average latency of 0.77s.

In summary, the contributions of our paper are five-fold:

- We propose pVoxel, the first voxel based point cloud analysis that realizes generic FP reduction for traffic detection.
- We utilize voxels, which are extended from the 3D physical space, to enable point cloud analysis in the high-dimensional traffic feature space for efficient FP processing.
- We develop a community based density analysis algorithm that represents adjacent voxels as communities and identifies FPs by learning density features.
- We develop a theoretical analysis framework established by the stochastic geometry to prove the effectiveness of pVoxel.
- We prototype pVoxel with CUDA and use the extensive experiments with various real-world FPs to validate its accuracy and efficiency.

We release the source code of pVoxel [66]. The rest of the paper is organized as follows: Section 2 presents problem statement and the threat model. Section 3 presents the motivation and the high-level design of pVoxel. In section 4, we describe the detailed designs. In Section 5, we conduct the theoretical analysis. In Section 6, we experimentally evaluate the performances. Section 7 discusses the practicality of pVoxel. Section 8 reviews related works. In Section 9, we conclude this paper.

2 PROBLEM STATEMENT & THREAT MODEL

2.1 Problem Statement

The goal of the paper is to identify FPs without any prior knowledge, which is different from the existing methods (i.e., retraining [25] and whitelists [80]) that use manually identified FPs to reduce similar foreseeable FPs. In particular, manually identifying even a small portion of FPs raised by traffic detection systems will consume huge manual labor due to the large number of FPs. We establish benchmarks by using Random Forest and SVM to classify the benign traffic in MAWI backbone network traffic datasets [87] (Jan. 2020) and NTP amplification attack traffic [27] according to the extracted CICFlowMeter features [16], which achieve over 99.93% precision and recall. Figure 1 depicts the number of FPs and FPR. We observe
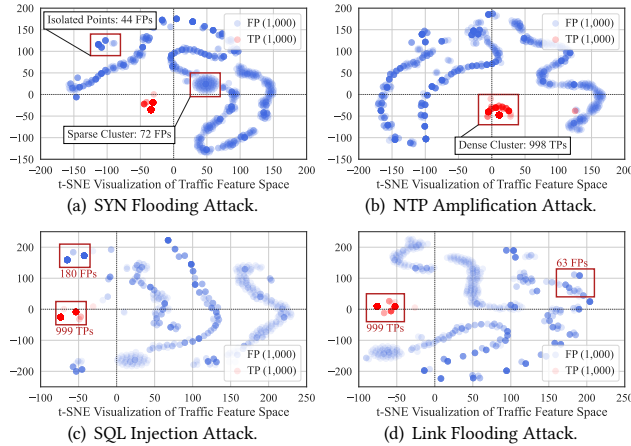
Figure 2: Scatter of TPs and FPs from different datasets.

that due to the large number of flows (806.43M flows per day [87]), a 0.19% negligible FPR that is lower than state-of-the-art methods [8, 31, 59, 97] will result in 1.54 million FPs per day. It is obvious that a small number of experts cannot cope with such massive FPs.

To achieve practical FP identification for different ML based detection systems in the real world, we develop pVoxel achieving the following goals.

(1) **Black-Box Detection.** pVoxel should not obtain any information about ML based traffic detection systems, including ML algorithms, model parameters, and training sets, which is a realistic setting for many commercial closed-source security tools, e.g., Cisco ETA [19]. Notably, traditional retraining methods are impractical under this restriction as they add the manually identified FPs into training sets and retrain the ML models based on original parameters [25].

(2) **Unknown Testing Sets.** pVoxel should not obtain any prior knowledge on testing sets, for instance, the benign IP addresses. Note that, it is infeasible to apply the traditional whitelist methods under this restriction, which verify if IP addresses that frequently trigger FPs are held by legitimate users (e.g., employees of Apple Inc.) [80] or reputable ASes [11], and then apply IP filtering rules to constrain the FPs. Besides, IP whitelists can be exploited to launch evasion attacks by IP spoofing [29].

In the nutshell, pVoxel should identify FPs only according to the feature vectors associated with the detected traffic and be applicable to various detection methods. Thus, it can identify FPs raised in testing phases without interference with model training though it can be integrated into training pipelines of detection methods. Moreover, pVoxel should realize efficient FP identification with low latency. These issues cannot be addressed by the existing methods [11, 24, 25, 80].

2.2 Threat Model

This paper aims to develop an FP identification method for ML based malicious traffic detection (i.e., pVoxel) that can be integrated into the existing Security Information and Event Management (SIEM) systems [3], which are widely deployed in Security Operation Centers (SOCs) to manage and visualize the alarms produced by various security tools [1] (e.g., host and network based IDSes [24, 31, 59]).

Similar to the threat models in the existing traffic detection studies [8, 31, 32, 59], we consider that an SOC operates various ML based malicious traffic detection systems according to the Internet traffic replicated by different routers via port mirroring (e.g., Cisco SPAN [20]). Upon receiving the replicated traffic, these systems extract one traffic feature vector for one flow (e.g., flow completion time [97]) or for one packet (e.g., packet length [39]), and utilize ML models to classify whether the feature vectors denote benign or malicious traffic [11, 39, 97]. Once detection systems classify feature vectors as malicious ones, they send the feature vectors to the SIEM to trigger alarms that may be either TPs or FPs [1, 48].

The FP identification method takes a batch of the alarms as input, for instance, the alarms within 20 seconds, and classifies each alarm as TP or FP according to the associated feature vectors. Finally, it assigns high priority to the TPs through the SIEM that allows SOC operators to instantly respond to attack traffic [83]. Moreover, similar to the existing studies [13, 29, 32], we consider that attackers may launch evasion attacks (e.g., IP spoofing [55, 62]).

3 DESIGNS OF PVOXEL

3.1 Key Observations

In Section 2.1, we show that it is infeasible for experts to identify FPs by analyzing individual alarms. However, it is still possible to identify the FPs by utilizing the relationship among the alarms. Specifically, we observe that feature vectors associated with TPs exhibit dense distributions, while the ones associated with FPs exhibit sparse distributions. Figure 2 shows the scatter plots corresponding to randomly sampled flow feature vectors associated with equal amounts of FPs and TPs. We observe that FPs are mainly located in the low-density regions that contain sparse clusters and isolated points, since diverse benign user behaviors generate various flows with different flow features. In contrast, TPs are represented by dense clusters in the high-density regions because malicious flows are normally generated by tools [26, 27, 29, 30, 43] that generate many flows with similar flow features [13, 29]. Thus, we can classify alarms into FPs and TPs according to the distinct density features in the traffic feature space.

3.2 Overview of pVoxel

In this paper, we propose pVoxel that identifies FPs by utilizing the point cloud analysis methods [67] that analyze the topological features among massive points for various tasks, e.g., 3D object detection [67], rendering [22], and space segmentation [94]. Specifically, given that the numbers of traffic feature vectors (i.e., points) are large, we design an efficient voxel based analysis method [53] that processes the vectors within a high-dimensional cube as a unit (named voxel) to reduce the processing overhead [34]. We extend voxels [34, 53, 67] in the 3D space to the high-dimensional traffic feature space. Thus, we can identify the voxels located in the high-density regions as TPs and the voxels in the low-density regions are detected as FPs.

Figure 3 shows the architecture of pVoxel. It consists of three modules to achieve the point cloud analysis for identifying FPs, i.e., voxel construction, community construction, and density analysis. Note that, pVoxel can be applied to detection systems built upon different types of features, e.g, flow-level [8, 75] or packets-level [39,
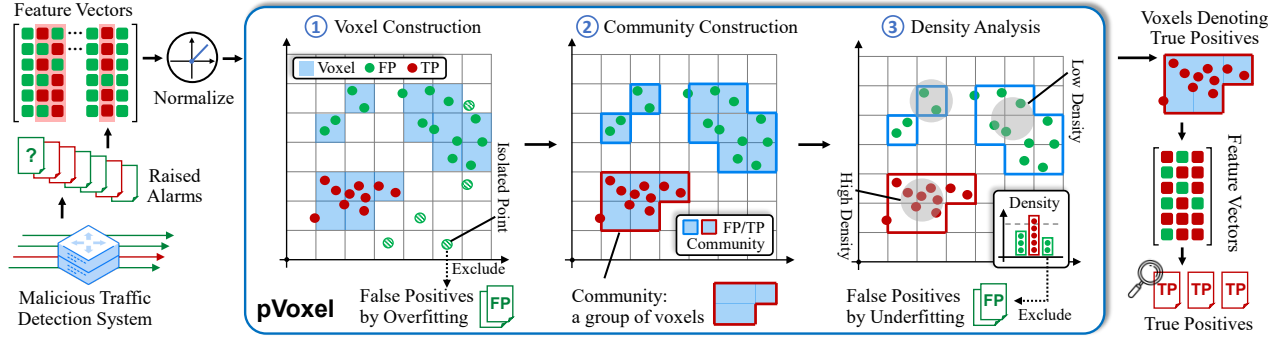
Figure 3: The overview of pVoxel.

59] features. For simplicity, we describe our design by taking flow-level traffic features based detection as an example.

Voxel Construction. pVoxel normalizes flow-level feature vectors associated with alarms and transforms the vectors into the points in a bounded feature space. It represents the points within a high-dimensional cube as a voxel. Meanwhile, by excluding the voxels that represent few points, it identifies isolated points in the low-density regions, which denote the FPs incurred by overfitting, i.e., the decision boundary is too close to certain types of benign traffic that makes drifted patterns misclassified. Note that, a voxel can represent many points, which ensures high processing efficiency. We will detail how pVoxel constructs voxels in Section 4.1.

Community Construction. pVoxel classifies adjacent voxels into a community to capture the topological features among the voxels. We define two voxels as adjacent ones according to both Manhattan distances and the length of the feature vectors (i.e., the dimension of the feature space) so that pVoxel can be adapted to various methods using different lengths of traffic features [8, 16, 75, 97]. Note that, we analyze the communities instead of the individual voxels to further improve the processing efficiency. We will describe the details of the community construction in Section 4.2.

Density Analysis. Now we extract four density features for each community and utilize unsupervised clustering to detect the communities with significantly high densities that mainly denote TPs. The rest of the communities represent the sparse clusters in the low-density regions that denote the FPs incurred by underfitting, i.e., certain types of benign traffic cannot be correctly classified by ML models. Note that, the unsupervised clustering method does not require any manually identified FPs for training and thus enables detecting unforeseen FP alarms. We will detail the density analysis in Section 4.3.

Note that, pVoxel can identify FPs generated by a broad spectrum of methods using various traffic features. We can process the features other than flow-level features so that pVoxel can be adapted to these methods. The details can be found in Section 4.4.

4 DESIGN DETAILS

In this section, we present the design details of pVoxel. For simplicity, we describe our design where we use flow-level traffic features based detection as an instance, which is most commonly used [8, 9, 75, 88, 96, 97], and then present how pVoxel can be adapted to methods with other types of traffic features. We will present a real-world case study in Section 7.3.

4.1 Voxel Construction

pVoxel normalizes flow-level features associated with alarms and processes the features as points in the traffic feature space, where points are represented by high-dimensional cubes (i.e., voxels). We extend Bartos *et al.* [9] to develop a mathematical representation. Let N denote the number of feature vectors which equals the number of alarms raised by flow-level feature based methods. We use $\vec{s}_i = [s_{i1}, \ldots, s_{iM}]^T (1 \leq i \leq N)$ and M to indicate the feature vector associated with the ith alarm and the length of the feature vector, respectively. Note that, methods using different numbers of flow-level features have different values of M [88, 89, 97]. Let matrix S denote the flow-level features of all alarms, where s_{ij} is defined as the ith alarm's j^{th} flow-level feature, e.g., the flow completion time (FCT) [16] and the number of packets in a flow [97]:

$$S = [\vec{s}_1, \ldots, \vec{s}_i, \ldots, \vec{s}_N] = \begin{bmatrix} s_{11} & \cdots & s_{N1} \\ \vdots & \ddots & \vdots \\ s_{1M} & \cdots & s_{NM} \end{bmatrix}. \quad (1)$$

To make the flow-level features to be numerically stable, we perform a logarithmic transformation to reduce the range of the feature and prevent arithmetic overflow during the point cloud analysis. We use $Q = \log_2(1 + S)$ to indicate the result. Then, we transform each element of $Q = [q_{ij}](1 \leq i \leq N, 1 \leq j \leq M)$ to $[0, 1]$ by performing min-max normalization for each of the M rows of matrix Q. Now, we obtain normalized feature vectors which are represented by N points $\mathcal{P} = [\vec{p}_1, \ldots, \vec{p}_N]$:

$$\begin{cases} \vec{\alpha} = [\alpha_j], & \alpha_j = \min(q_{1j}, \ldots, q_{ij}, \ldots, q_{Nj}), \\ \vec{\beta} = [\beta_j], & \beta_j = \max(q_{1j}, \ldots, q_{ij}, \ldots, q_{Nj}), \\ 1 \leq j \leq M, & 1 \leq i \leq N, \end{cases} \quad (2)$$

$$\vec{p}_i = [p_{i1}, \ldots, p_{ij}, \ldots, p_{iM}]^T, \quad p_{ij} = \frac{q_{ij} - \alpha_j}{\beta_j - \alpha_j}. \quad (3)$$

We define a voxel $\mathcal{V}(\vec{a})$ as an M-dimensional small cube with side length ϵ in the normalized traffic feature space $\mathfrak{F} = [0, 1]^M$, where $\vec{a} = [a_1, \ldots, a_M]^T$ is the index of the voxel. Thus, the space covered by the voxel can be represented by the Cartesian product:

$$\mathcal{V}(\vec{a}) = [\epsilon(a_1 - 1), \epsilon a_1] \times \cdots \times [\epsilon(a_M - 1), \epsilon a_M], \quad (4)$$

$$\forall a_j \in \{1, \ldots, \lceil 1/\epsilon \rceil\}, \quad (1 \leq j \leq M). \quad (5)$$

Moreover, it is clear that $\mathcal{V}(\vec{a}) \in \mathfrak{F}$ holds for $\forall \vec{a}$, and the number of distinct voxels equals $H = \lceil 1/\epsilon \rceil^M$.

We define function $\xi(i, \vec{a}; \mathcal{P})$ that judges if point $\vec{p}_i$ is within the space covered by a voxel $\mathcal{V}(\vec{a})$:

$$\xi(i, \vec{a}; \mathcal{P}) = \begin{cases} \{i\}, & \text{if} \quad \forall 1 \le j \le M, \quad \mathrm{p}_{ij} \in [\epsilon(\mathrm{a}_j - 1), \epsilon \mathrm{a}_j), \\ \phi, & \text{else.} \end{cases} \quad (6)$$

We aggregate the points $\mathcal{P}$ into voxels that are indexed by the vectors in the universal set of the index vectors $\mathcal{A} = \{\vec{a}_1, \ldots, \vec{a}_H\}$. Then, we define function $\zeta(\vec{a}; P)$ that outputs the indices of the points that are within the voxel indexed by $\vec{a}$:

$$\zeta(\vec{a}; P) = \bigcup_{i=1}^{N} \xi(i, \vec{a}; \mathcal{P}). \quad (7)$$

Moreover, we exclude the voxels that represent lower than E points (see Table 7 for all hyper-parameters). We define the indices of retained voxels as $\mathcal{A}^* = [\vec{a}_1^*, \ldots, \vec{a}_U^*]$, which satisfies $\forall \vec{a}^* \in \mathcal{A}^*$, $|\zeta(\vec{a}^*; P)| \ge E$, where U is the number of retained voxels.

The reason why we exclude voxels representing few points is that these points are mainly isolated points which denote drifted flow patterns (see Figure 5(a)). Such drifted patterns are easily misclassified as FPs by an overfitted model (see Figure 4). That is, the decision boundary of such a model is too close to the patterns that frequently appear in the training set (e.g., mail traffic), and leads to the misclassification of drifted traffic patterns, for example, the traffic delivering mail with many huge attachments (see the upper left corner of Figure 2(a)).

Note that, isolated points associated with FPs are sparsely distributed, which only have a small number of neighbors (see Figure 2(a)), so that these isolated points can be excluded by applying one threshold on the density of points. By excluding isolated points, we significantly reduce the number of points, which can effectively reduce 94.89% processing latency of pVoxel.

4.2 Community Construction

We use communities to represent the retained adjacent voxels. First, we define the function $\mathrm{N}(\vec{a}_i, \vec{a}_j; D, M)$ that judges if two voxels indexed by $\vec{a}_i$ and $\vec{a}_j$ are adjacent in the feature space:

$$\mathrm{N}(\vec{a}_i, \vec{a}_j; D, M) = \begin{cases} 1, & \text{if} \quad d_{\text{Manhattan}}(\vec{a}_i, \vec{a}_j) \le DM, \\ 0, & \text{else,} \end{cases} \quad (8)$$

where D is a predefined hyper-parameter and $d_{\text{Manhattan}}$ calculates the Manhattan distance between a pair of points. Note that, the adjacency relationship is defined according to both Manhattan distances and the dimension of the feature space (M). Since we aim to adapt pVoxel to various ML based traffic detection systems that use different numbers of features, e.g., NetBeacon [97] uses 13 features ($M = 13$), and N3IC [75] uses 17 features ($M = 21$).

We use the Floyd-Warshall algorithm to obtain the reachability between $U \times U$ pairs of voxels. And we use a community to represent a group of voxels that constitute a strongly connected component, i.e., each voxel is reachable to any other ones. Specifically, we use a set $C = \{\vec{a}_i, \ldots, \vec{a}_{|C|}\}$ to indicate a community that contains $|C|$ voxels and satisfies the following two requirements:

$$|C| \times |C| = \sum_{i=1}^{|C|} \sum_{j=1}^{|C|} \mathrm{N}(\vec{a}_i, \vec{a}_j; D, M), \quad (9)$$

$$\forall \vec{a}_i, \vec{a}_j \in \mathcal{A}^*, \quad (1 \le i \le |C|, 1 \le j \le |C|). \quad (10)$$

According to the definition of the strongly connected component, $\cap_{i=1}^{G} C_i = \phi$ and $\cup_{i=1}^{G} C_i = \mathcal{A}^*$ hold for G identified communities, which ensures that each voxel is represented by exactly one community. Note that, pVoxel does not analyze the features of voxels directly. Instead, it analyzes the features of communities to further reduce the overhead and ensure efficient FP identification.

4.3 Density Analysis

In this module, we extract features of each constructed communities for the FP community detection. According to Figure 2, the communities denoting FPs are located in the regions with sparsely distributed points. To identify the communities denoting FPs with the low densities, we extract four point density features, i.e., the number of points represented by a community, and the average number of points within the balls centered by the voxels represented by a community. Formally, we use $\mathrm{f_D}^{(k)}(C_i)$, $k \in \{1, 2, 3, 4\}$ to indicate the extracted features for the ith community:

$$\mathrm{f_D}^{(1)}(C_i) = \left| \bigcup_{\vec{a}}^{C_i} \zeta(\vec{a}; \mathcal{P}) \right|. \quad (11)$$

We define function $\Omega(\vec{a}, r; P)$ that outputs the points within the ball centered by voxel $\mathcal{V}(\vec{a})$ with radius r:

$$\omega(i, \vec{a}, r; \mathcal{P}) = \begin{cases} \{i\}, & \text{if} \quad \left\| \vec{p}_i - \epsilon \vec{a} \right\|_2 \le r, \\ \phi, & \text{else.} \end{cases} \quad (12)$$

$$\Omega(\vec{a}, r; \mathcal{P}) = \bigcup_{i=1}^{N} \omega(i, \vec{a}, r; \mathcal{P}). \quad (13)$$

$$\mathrm{f_D}^{(k)}(C_i) = \frac{1}{|C_i|} \left| \bigcup_{\vec{a}}^{C_i} \Omega(\vec{a}, r^{(k-1)}; \mathcal{P}) \right| \quad (k \in \{2, 3, 4\}). \quad (14)$$

We apply K-Means unsupervised clustering to detect the communities with significantly high densities which denote TPs (see Figure 5(b)); while the rest of the communities are located in the low-density regions which denote FPs. Given that the number of communities is already low, we empirically set the K-value for K-Means to one. Finally, pVoxel outputs the indices of the points which are represented by the communities with low densities:

$$\mathcal{R}_C = \{i | \|\mathrm{f_D}(C_i) - \vec{c}\|_2 \le d_{\text{Center}}, 1 \le i \le G\}, \quad (15)$$

$$\mathcal{R}^* = \bigcup_{i}^{\mathcal{R}_C} \bigcup_{\vec{a}}^{C_i} \zeta(\vec{a}; P), \quad (16)$$

where $\vec{c}$ is the clustering center, $\mathcal{R}_C$ is the indices of identified communities denoting FPs, and $\mathcal{R}^*$ is the indices of flow features associated with identified FPs. Note that, these FPs are mainly caused by underfitting, which means certain types of benign traffic cannot be correctly classified. The features of such misclassified benign traffic are represented by sparse clusters in the low-density regions, which can be easily captured by our density analysis.

4.4 Adapting pVoxel to Various Methods

pVoxel can be adapted to detection methods using features other than flow-level features. We utilize a preprocessing step to process these features:
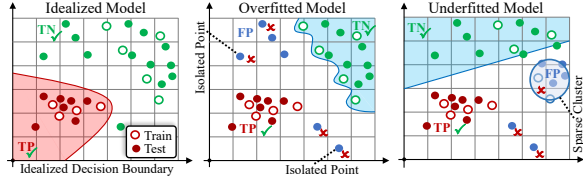
Figure 4: FPs incurred by overfitting and underfitting.

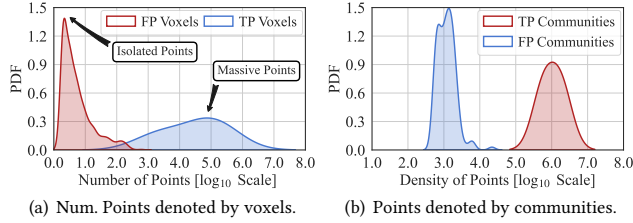(a) Num. Points denoted by voxels. (b) Points denoted by communities.

Figure 5: Points denoted by voxels and communities.

Packet-Level Features. The packet based methods extract one feature vector for one packet [39, 59], instead of one flow. Consequently, they produce more alarms by raising one alarm for each packet. Therefore, handling the massive per-packet alarms incurs high performance penalty. To tackle this issue, pVoxel converts the per-packet alarms to per-flow ones. Specifically, it merges the alarms triggered by the packets from the same flow into one alarm which is associated with a flow-level features vector calculated according to the packet-level features. For example, the flow size feature [97] can be calculated by accumulating the length of each packet. In practice, we calculate the flow-level features used by N3IC [75].

Frequency Domain Features. The methods utilizing the frequency domain feature [31, 33] extract a feature matrix for a flow instead of a vector. Therefore, we transform each row of the matrix into a scalar by calculating the average of the row, which equals the power of the corresponding frequency component [31]. By this way, we convert the metrics to vectors, and thus pVoxel can handle the FPs using the same way as flow-level features based detection.

Graph Based Features. The existing methods utilize graphs to represent the traffic interaction patterns among attackers and victims [32, 72]. These methods use edges to denote flows and learn features of edges for attack detection [32, 47]. pVoxel identifies FPs generated by these methods according to edge feature vectors that denote flows. Specifically, to construct a feature vector for a flow, we extract the in- and out-degree for the source and destination of the corresponding edge associated with the flow. Thereafter, pVoxel processes the feature vectors using the same way as flow-level features.

5 THEORETICAL ANALYSIS

In this section, we develop a *stochastic geometry model* to prove that pVoxel can efficiently identify FPs by detecting the voxels located in the low-density regions, which represent isolated points and sparse clusters. Meanwhile, pVoxel ensures that TPs are accurately retained by detecting voxels located in the high-density regions. The proofs can be found in Appendix A.

5.1 Density Modeling for Traffic Feature Space

First, we establish the *density model of the traffic feature space*, a theoretical analysis framework that aims to analyze the density of traffic feature vectors represented by a voxel, from the perspective of stochastic geometric [21, 77]. In general, our theoretical analysis is based on the following key idea: (i) we model packet sequences (i.e., benign and malicious flows) as random variable sequences; (ii) we model flow feature extraction methods as the functions of the random variables, and obtain the geometric distribution of the points which denote the extracted features; (iii) we calculate the expected density of the points represented by one voxel to prove the effectiveness of pVoxel.

In the framework, we consider N flows that are represented by N sequences of observable packet-level features (e.g., packet length and arrival interval). Specifically, we use a random variable vector $\vec{s}_i = [s_{i1}, \ldots, s_{iL}]^{\mathsf{T}}$ ($1 \leq i \leq N$) to indicate a sequence, where L is a random variable denoting the length of the sequence which equals to the number of packets in a flow. According to existing studies [31, 32], each element in $\vec{s}_i$ obeys normal distribution, i.e, $s_{ij} \sim \mathcal{N}(\mu, \sigma^2)$ holds for $\forall\, 1 \leq i \leq N$ and $\forall\, 1 \leq j \leq L$. Without loss of generality, we assume that the packet-level features are normalized before extracting the flow-level features, i.e., $\mu = 0$.

We denote a flow-level feature extraction method as a vector-valued function $\mathsf{f_E}$ that transforms a random variable vector $\vec{s}_i$ into a single random variable v_i, for example, $\mathsf{f_E}$ can accumulate the length of each packet to derive the size of a flow [97]. In this paper, we consider five typical feature extraction methods that are widely used [8, 11, 31, 80, 89, 96], including: (i) $\mathsf{Sum}(\cdot)$: accumulative features; (ii) $\mathsf{Avg}(\cdot)$: average based feature; (iii) $\mathsf{Min}(\cdot)$: minimum value based feature; (iv) $\mathsf{Range}(\cdot)$: range based feature; (v) $\mathsf{Var}(\cdot)$: variance based feature. For simplicity of notation, we analyze one of these flow features each time, which is one dimension of the entire traffic space. Thus, the N flow features $\vec{v} = [v_1, \ldots, v_N]$ are denoted by the points randomly distributed on an infinite line.

Here, we define the following two geometric features of the points (see Figure 6) which will show the effectiveness of pVoxel:

Definition 5.1. (Range of traffic features.) Let R indicate the distance between the points representing the maximum and minimum of the flow features, which means $R = \max(\vec{v}) - \min(\vec{v})$.

Definition 5.2. (Density of traffic features.) Let D indicate the density of the points within the range ($R > 0$), i.e., $D = \frac{N}{R}$.

Given that we analyze a single dimension of the entire traffic feature space, voxels are denoted by the intervals of length ϵ on the infinite line with random positions. It implies that the expected density of all the non-empty intervals is equal to the expected density of the whole range R. Therefore, we can use $\mathrm{E}[D]$ to indicate the expected density of the voxels.

5.2 Stochastic Geometry Analysis

In this section, we analyze the expected densities for N benign flows (FPs) and N malicious flows (TPs) which are indicated by $\mathrm{E}[D_{\mathrm{B}}]$ and $\mathrm{E}[D_{\mathrm{M}}]$, respectively. Note that, the variance parameter of benign flows is significantly larger than that of malicious flows, i.e., $\sigma_{\mathrm{B}} > \sigma_{\mathrm{M}}$, since benign flows are triggered by diverse user behaviors with many complex patterns. Similarly, L_{B} is larger than L_{M} according to

the observation that many malicious flows are short flows [28, 32, 49]. Our empirical studies using the same configuration as Figure 2 show that $\sigma_B/\sigma_M \approx 30.21$ and $L_B/L_M \approx 160.03$. Eventually, we derive the following theorems from the analysis:

THEOREM 5.3. *(Ratio of densities for accumulative features.) When* $f_E(\cdot) = \text{Sum}(\cdot)$, *the ratio of the expected point densities associated with the voxels denoting FPs and TPs is:*

$$\frac{E[D_B]}{E[D_M]} = \left(\frac{\sigma_B}{\sigma_M}\right)^{-1} \left(\frac{\sqrt{L_B}}{\sqrt{L_M}}\right)^{-1}. \tag{17}$$

THEOREM 5.4. *(Ratio of densities for average based features.) When* $f_E(\cdot) = \text{Avg}(\cdot)$ *and* L_B/L_M *is a constant, the ratio of the expected point densities associated with the voxels denoting FPs and TPs is inversely proportional to* σ_B/σ_M.

According to the above theorems and the inequalities of σ and L, we conclude that FPs triggered by benign flows are represented by the voxels with low density features. Thus, pVoxel can effectively exclude FPs by identifying the voxels representing either isolated points (Section 4.1) or sparse clusters (Section 4.3) in the low-density regions. In contrast, TPs triggered by malicious flows will be retained by pVoxel according to the voxels denoting high-density clusters. In the same way, we can draw similar conclusions for the other traffic features.

THEOREM 5.5. *(Upper bound of the expected density for range based features.) When* $f_E(\cdot) = \text{Range}(\cdot)$, *an upper bound of the expected point density of the voxels is:*

$$E[D] \leq \frac{NL-1}{2\sigma L} \sqrt{W_0\left(\frac{N^2L^2-1}{2\pi}\right)^{-1}}, \tag{18}$$

where $W_0(\cdot)$ *is the first Lambert W function (i.e.,* $y = W_0(x)$ *iff.* $x > -\frac{1}{e}$ *and* $ye^y = x$).

Moreover, we prove that the bound is inversely proportional to σ and the upper bound is monotonically decreasing over the range $L \geq 1$. Thus, the higher variance σ_B and the larger length L_B lead to lower bounded densities for the voxels denoting FPs. In addition, the conclusion holds for minimum value based features:

THEOREM 5.6. *(Upper bound of the expected density for minimum value based features.) When* $f_E(\cdot) = \text{Min}(\cdot)$, *a lower bound of the expected range of the features is:*

$$E[R] \geq \frac{2\sigma NL}{NL-1} \sqrt{W_0\left(\frac{N^2L^2-1}{2\pi}\right)} - \frac{2\sigma L}{L-1} \sqrt{W_0\left(\frac{L^2-1}{2\pi}\right)} - o(0.12\sigma), \tag{19}$$

and an upper bound of the point density can be derived according to Definition 5.2, i.e., $E[D] = N/E[R]$.

We can prove a similar theorem for the maximum based features by utilizing the symmetry of the range. Due to the page limits, we omit the proof. Finally, we analyze the density for variance based features under the condition that L is large and small, separately.

THEOREM 5.7. *(Lower bound of the expected density for variance based features when flow length is above 50.) When* $f_E(\cdot) = \text{Var}(\cdot)$ *and* $L > 50$, *an upper bound of the expected range of the features is:*

$$E[R] \leq \frac{N\sigma^2}{e\sqrt{\pi L}} + \frac{\sigma^2}{2}(1 - \text{erf}(1)), \tag{20}$$
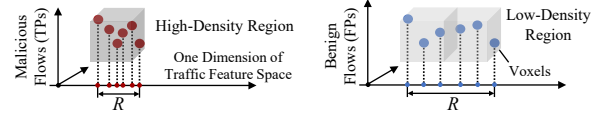
Figure 6: Defined range and density geometric features.

where $\text{erf}(\cdot)$ *is the error function:* $\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} \mathrm{d}t$. *Now, we can obtain a lower bound of the density according to Definition 5.2.*

THEOREM 5.8. *(The expected density for variance based features when the flow length equals two.) When* $f_E(\cdot) = \text{Var}(\cdot)$ *and* $L = 2$, *the expected range of the flow features is:*

$$E[R] = \frac{N\sigma^2}{N+1} \sum_{i=1}^{N+1} \frac{1}{i} - \frac{\sigma^2}{N}. \tag{21}$$

Similarly, the expected density can be calculated by $E[D] = N/E[R]$.

Note that, $E[D]$ is inversely proportional to σ^2. It means that TPs triggered by malicious flow with the lower variance σ_M are denoted by the voxels with significantly high densities, which can be easily retained by pVoxel.

In summary, for all the types of traffic features, we prove that the points denoting TPs exhibit dense distribution (i.e., higher $E[D]$), since the attack traffic that triggers the TPs is generated by attack tools [13, 29] which construct many similar short flows (i.e., lower L and σ) [44, 49, 69]. Thus, pVoxel can easily detect the points denoting TPs in the high-density regions (see Section 4.3). Meanwhile, we also prove that the points denoting FPs exhibit sparse distribution (i.e., lower $E[D]$), since diverse benign user behaviors generate many long flows with various flow features (i.e., higher L and σ) [28, 32]. Therefore, pVoxel captures such traffic features in the low-density regions by detecting isolated points and communities with low density features. These theoretical analysis conclusions are consistent with our real-world empirical studies (see Section 3.1) which lay the foundations of our effective FP identification.

6 EXPERIMENTAL EVALUATION

In this section, we prototype pVoxel and evaluate its performance by identifying FPs raised by 11 state-of-the-art methods under 75 real-world attacks. In particular, the experiments will show that pVoxel is able to:

(1) effectively identify FPs raised by various methods on different datasets while improving many accuracy metrics (Section 6.2).
(2) be robust to various ML models with different hyper-parameter settings (Section 6.3).
(3) outperform traditional retraining based methods which require manually identified FPs (Section 6.4).
(4) achieve high processing throughput with low latency (Section 6.5).
(5) achieve high robustness under evasion attacks (Section 6.6).

6.1 Experiment Setup

Implementation. We prototype pVoxel with more than 5,700 Lines of Code (LOC), which is compiled by GCC 9.4.0, NVIDIA CUDA 11.4.48 compiler (nvcc) through ninja 1.10.0 and cmake 3.16.3. Specifically, we use CUDA C++ API [63] to implement the voxel construction module and the Floyd-Warshall algorithm for the community

Table 2: Basic properties of detection methods and statistics of datasets.

Methods	Traffic Feature	ML Model (Unsupervised)	Number of Raised FPs on Different Datasets (Num. FPs/s)									Accuracy Metrics[1]		
			HyperVision Datasets			CIC Datasets		Kitsune	Whisper	NetBea.	All	AUC	ACC.	FPR
			Brute	LowRate	Advanced	IDS2017	DoS2019							
CICFlowMeter*	Flow-level	Random Forest	242.65	827.68	227.78	148.92	168.16	63.42	228.99	225.46	266.63	0.9809	0.9844	*0.0378*
FlowLens	Flow-level	Decision Tree	416.77	887.56	222.68	179.50	403.13	436.43	412.56	216.81	396.93	0.9840	0.9886	0.0304
Jaqen*	Flow-level	Random Forest	400.11	545.89	75.28	8.69	877.24	43.29	83.62	280.82	289.37	0.9856	0.9822	0.0270
N3IC	Flow-level	Binary Neural Net.	239.74	724.22	21.30	30.21	19.51	38.77	34.92	126.79	154.43	*0.9610*	*0.9714*	0.0173
NetBeacon	Flow-level	Decision Tree	335.13	855.20	25.27	171.49	0.24	29.17	54.36	186.50	207.17	0.9938	0.9936	0.0113
nPrintML	Packet-level	AutoML	52.40	124.60	0.77	4.83	1.51	3.52	25.55	57.99	33.90	*0.9958*	*0.9990*	*0.0005*
HyperVision	Graph	K-Means/DBSCAN	41.36	5.12	4.39	2.00	4.00	106.04	94.67	0.59	32.27	0.9999	0.9969	0.0002
FAE	Frequency	Autoencoder	30.43	12.34	18.91	26.46	33.18	10.29	29.17	25.36	23.26	0.9113	0.9919	0.0072
FSC	Flow-level	K-Means	821.59	1504.19	315.19	555.88	605.69	203.96	467.96	429.15	612.95	0.9151	0.9293	0.0798
Kitsune	Packet-level	Autoencoder	2294.18	2204.17	N/A[2]	1024.04	1223.91	1142.14	1092.22	1785.05	1537.96	0.8127	0.8981	0.1417
Whisper	Frequency	K-Means	12.06	9.17	76.76	12.66	11.82	7.40	26.56	17.40	21.73	0.9135	0.9939	0.0057
Average	/	/	444.22	700.01	98.83	196.79	304.40	189.49	231.87	304.72	325.15	0.9503	0.9754	0.0326

[1] The highest accuracy is calculated by using per-flow labels as ground truth. We highlight best ● and worst ● cases for <u>unsupervised</u> and *supervised* methods.
[2] Kitsune is not designed to detect the attacks constructed with various encrypted traffic.

construction module. Moreover, we use mlpack [60] (version 3.4.2) to implement the K-Means for the density analysis module.

Testbed. We deploy pVoxel on a testbed built upon a DELL PowerEdge server with an Intel Xeon E2699 v4 CPU, 4 Tesla V100 GPUs (one is used, driver version 470.103.01), 512GB DDR4 memory (32GB is used), Intel I350 NIC (1Gb/s), and Ubuntu 18.04.6 (Linux 4.15.0) with Docker 20.10.13. Moreover, we connect the server to another similar machine that replays traffic datasets to trigger alarms.

Datasets. In the experiments, we collect alarms raised by 11 state-of-the-art methods under 75 real-world attacks in 8 different networks. We summarize the statistics of the datasets in Table 2. Specifically, these methods extract flow features [8, 16, 31, 54, 75, 97], packet features [39, 59], the frequency domain features [31, 33], and graph features [32, 47] and utilize various ML algorithms with different settings. We deploy the open-source methods [31, 32, 59] without any modification and retrain their ML models. Meanwhile, we implement closed-source methods [33, 54] and the methods relying on specific hardware [75, 97]. Note that, to achieve end-to-end detection, we implement ten ML models to learn CICFlowMeter flow features [16] which is merely a feature set without a specific ML model [16]. We use CICFlowMeter* to denote random forest (RF) model based detection that achieves the highest accuracy. And the rest nine models are used as benchmarks for robustness analysis in Section 6.3. Besides, we use RFs to learn the traffic features extracted by Jaqen [54] which is originally a fixed rule based method to show that it is potentially feasible for pVoxel to identify FPs raised by traditional rule based detection.

To generate alarms, we deploy these methods to detect attacks in various real-world datasets that cover 75 typical attacks, including: (i) CIC-IDS2017 [18] and CIC-DDoS2019 [18] datasets which collect attack traffic from physical network testbeds; (ii) Kitsune datasets [59], with real-world attacks targeting IoT devices [36]; (iii) NetBeacon datasets [97], which are collected in a private cloud; (iv) Whisper datasets [31], with both volumetric and stealthy attacks [26, 44, 50]; (v) HyperVision datasets [32], which contain both traditional flooding attacks [27, 35, 49, 69] and various advanced attacks [29, 56, 57, 64]. Note that, we do not use the malware and web

attack traffic because most of the methods do not consider detecting such advanced attacks. We replay these datasets according to their original packet rates. In addition, we replay the datasets [17, 18, 59] along with MAWI backbone network datasets [87] as the background traffic. Table 2 shows that the number of FPs triggered by the attack traffic is ranging between 21.73 ∼ 1537.96 FPs per second. Notably, unsupervised methods raise more FPs than supervised ones [4, 76], while the frequency and graph based methods raise fewer FPs, since they raise one alarm for the flows with same IP addresses [32, 33].

Hyper-Parameter Selection. We conduct four-fold cross validation for pVoxel to prevent hyper-parameter bias [4]. In specific, the alarms are equally partitioned into four subsets. Each of the subsets is used once as a validation set to tune the hyper-parameters according to empirical studies. And the remaining three subsets are testing sets. Finally, four groups of results are averaged to produce the final results. Table 7 shows all the hyper-parameters.

Metrics. We mainly use reduced false positive rates (R.FPR) and numbers of reduced false positives (R.FPN) to evaluate the effectiveness of pVoxel. Meanwhile, we use reduced true positive rates (R.TPR) to show the collateral damages on TPs which should be low. Additionally, we measure the increase of various accuracy metrics that are widely used to evaluate ML based traffic detection systems [11, 24, 31, 39, 59, 79, 99], including the area under the precision-recall curve (AUPRC) and the receiver operating characteristic curve (AUROC), F1, accuracy (Acc.), precision (Pre.), equal error rate (EER), and Matthews correlation coefficient (MCC).

6.2 Accuracy Evaluation

Table 3 summarizes the accuracy of reducing FPs. In general, pVoxel can reduce 95.55% FPR (306.28 FPs/s) for the 11 state-of-the-art methods under the 75 attacks. By reducing FPs, pVoxel significantly improves detection accuracy of various methods.

First, we measure the performance of pVoxel by using R.FPR to show that pVoxel can effectively reduce FPs raised by various methods. From Table 3, we can see that pVoxel reduces 95.33% ∼

Table 3: Number and ratio of reduced FPs by pVoxel.

| Methods	HyperVision Datasets (46)[1]						CIC Datasets (2)				Kitsune (5)		Whisper (14)		NetBeacon (8)		Overall (75)	
	Brute		LowRate		Advanced		IDS2017		DoS2019									
	R.FPR	R.FPN	R.FPR	R.FPN	R.FPR	R.FPN	R.FPR	R.FPN	R.FPR	R.FPN	R.FPR	R.FPN	R.FPR	R.FPN	R.FPR	R.FPN	R.FPR	R.FPN
CICFlowMeter*	0.9786	237.97	0.9053	718.72	0.9473	214.54	0.9832	147.59	0.9261	155.73	0.9582	58.95	0.9601	215.29	0.9679	218.43	0.9533	245.90
FlowLens	0.9786	408.92	0.9999	887.56	0.9116	211.96	0.9909	179.19	0.9999	403.13	0.9999	436.43	0.9266	241.91	0.8876	172.48	0.9619	367.70
Jaqen*	0.9429	367.53	0.9999	545.89	0.8966	60.40	0.9999	8.69	0.9999	877.24	0.9990	43.19	0.9520	78.34	0.9621	263.42	0.9691	280.59
N3IC	0.9755	234.77	0.9999	724.22	0.9065	14.46	0.9999	30.21	0.9999	19.51	0.9987	38.69	0.9093	34.63	0.9653	119.76	0.9694	152.03
NetBeacon	0.9265	301.51	0.9999	855.20	0.8990	20.95	0.9948	171.33	0.9999	0.24	0.9999	29.17	0.9496	37.20	0.9735	178.58	0.9679	199.27
nPrintML	0.9251	38.84	0.9701	119.16	0.9333	0.53	0.9988	4.82	0.9999	1.51	0.9890	3.46	0.9469	16.98	0.8988	30.09	0.9578	26.92
Hypervision	0.9999	41.36	0.9999	5.12	0.9999	4.39	0.9999	2.00	0.9999	4.00	0.9999	106.04	0.9999	94.67	0.9999	0.59	0.9999	32.27
FAE	0.9221	28.28	0.9032	11.70	0.9071	17.67	0.9382	24.87	0.9605	31.87	0.9298	9.52	0.9147	26.95	0.9237	23.60	0.9249	21.81
FSC	0.8900	758.51	0.9480	1423.81	0.9146	286.68	0.9999	555.88	0.9999	605.69	0.8895	158.85	0.9069	399.62	0.9454	399.94	0.9368	573.62
Kitsune	0.9413	2163.95	0.8957	1928.37	N/A	N/A	0.9949	1019.53	0.9941	1216.69	0.9203	1063.84	0.8927	965.19	0.9999	1785.05	0.9484	1448.95
Whisper	0.9341	11.37	0.9243	8.48	0.9236	70.43	0.9295	11.77	0.9342	11.04	0.9025	6.68	0.9196	24.60	0.9023	15.80	0.9213	20.02
Average	0.9468	417.54	0.9588	657.11	0.9240	90.20	0.9846	195.99	0.9832	302.42	0.9625	177.71	0.9344	194.13	0.9479	291.61	0.9555	306.28

[1] The total number of separate datasets with different attacks.

96.94% FPR and 92.13% ∼ 99.99% FPR for supervised and unsupervised methods, respectively. Also, pVoxel can effectively reduce FPs raised by the methods using different traffic features. Specifically, pVoxel reduces 96.19%, 95.78%, 92.49%, and 99.99% FPs for FlowLens, nPrintML, FAE, and HyperVision which analyze flow, packet, frequency, and graph features, respectively. In addition, we also apply pVoxel to reduce FPs raised by EULER [47], i.e., a graph convolutional network based method that detects lateral movement behaviors of APTs [58]. As EULER is originally designed for log based detection, we adapt it into network traffic based detection [14]. We collect triggered FPs on six lateral movement traffic datasets [32]. pVoxel achieves similar performance of FP reduction. It reduces 94.45% ∼ 97.72% FPs.

Note that, we can draw similar conclusions by measuring the number of reduced FPs (R.FPN). Before applying pVoxel to reduce FPs, the 11 methods raise 325.15 FPs/s on average. The massive FPs are almost impossible for human experts to analyze. However, our experiments show that pVoxel can reduce 306.28 FPs/s on average. For instance, pVoxel can reduce 20.02 FPs/s for Whisper which raises 21.73 FPs/s and thus allow human experts to focus on true alarms (TPs). By this way, pVoxel significantly reduces labor costs for deploying ML based detection in real-world environments.

Second, we show that pVoxel can accurately identify FPs among the alarms triggered by the traffic from different datasets. In specific, pVoxel can reduces 94.79% ∼ 98.46% FPs on Kitsune datasets [59], NetBeacon datasets [97], and CIC datasets [17, 18] which mainly contain traditional attacks (e.g., amplification attacks [35] and network scanning [26]). Moreover, pVoxel can reduce 92.40% ∼ 95.88% FPs on HyperVision datasets [32] and Whisper datasets [31] that contain various advanced attacks (e.g., side-channel attacks [13, 29], link flooding attacks [46, 56], and password cracking attacks [10, 43]). Therefore, pVoxel achieves generic FPs identification for different datasets with various attacks.

Third, we measure the reduced TPR to show that pVoxel rarely misclassifies TPs as FPs. We present R.TPR which equals decreased recall in the last column of Table 4. We can see that reduced TPR is ranging between 1.58% ∼ 5.11% with an average of 2.62%, which
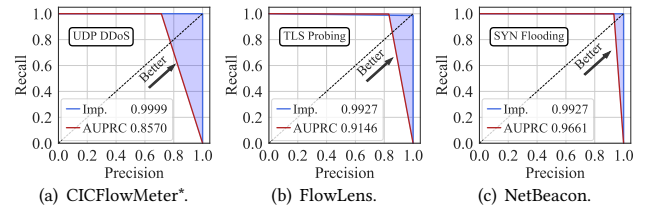
Figure 7: Examples of improved PRC by pVoxel.
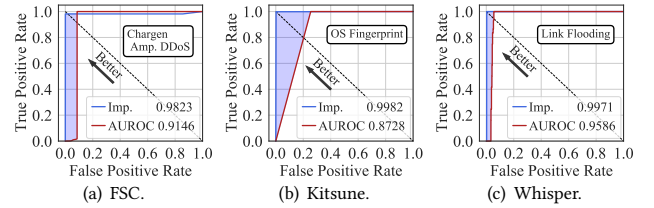
Figure 8: Examples of improved ROC by pVoxel.

can be bounded by 0.50% on 65% datasets. For instance, pVoxel misclassifies only 41, 72, 6, and 2 true positive alarms, which are raised by CICFlowMeter* when detecting HTTP probing [69], TLS vulnerability detection [57], host fingerprinting [27], and SQL injection attacks [29], respectively. Notably, the numbers of misclassified TPs are low, since the ratio of attack traffic that triggers TPs is significantly lower than the ratio of benign traffic, e.g., the ratio of malicious traffic is only 2.80% [32]. Besides, for the methods that frequently raise TPs, pVoxel still rarely misclassifies TPs. For example, pVoxel only misclassifies 44 TPs among the entire 107,162 TPs raised by FlowLens on CIC-DDoS2019 datasets.
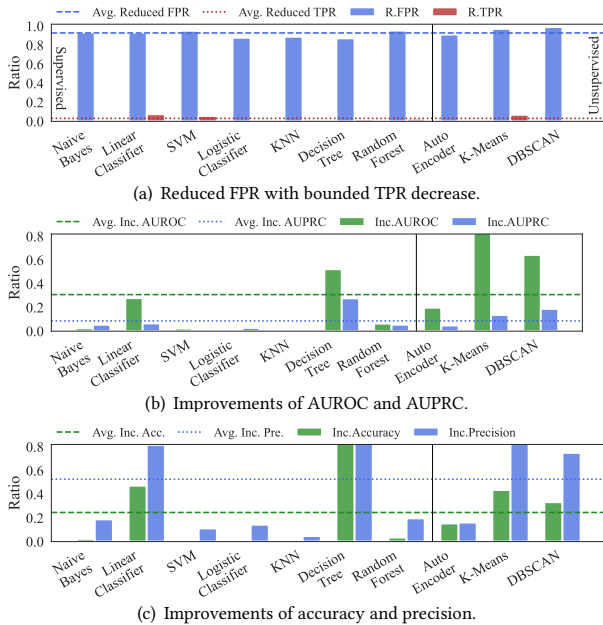
Finally, pVoxel can significantly improve various accuracy metrics by reducing FPs. We present the PRC and ROC in Figure 7 and Figure 8, where the area of the shadow equals increased AUPRC and AUROC. In particular, the improvements of AUPRC and AUROC over the original detection methods are ranging between 1.27% ∼ 33.66% and 0.43% ∼ 40.45%, respectively. Table 4 illustrates that pVoxel can improve five other accuracy metrics for various detection methods. By reducing FPR (see Table 3), pVoxel improves precision, accuracy, F1, and MCC by 44.72%, 5.73%, 22.85%, and 96.90%, respectively. Moreover, it can reduce 67.40% EER on average.

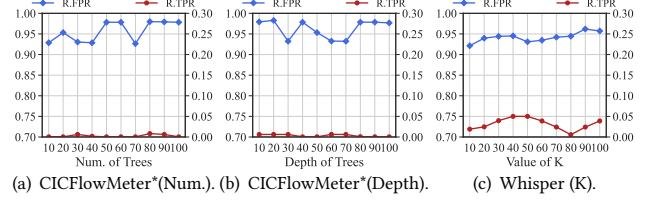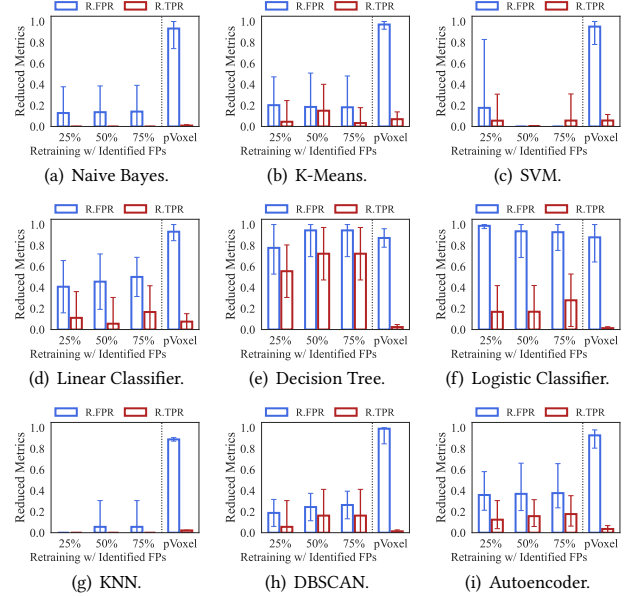**Table 4: Effects on various detection accuracy metrics.**

Methods	Increased Accuracy Metrics						Decreased.	
	PRC▲	ROC▲	Acc.▲	Per.▲	F1▲	MCC▲	EER▲	TPR▼
CICFlowMeter*	0.0545	0.0638	0.0344	0.1963	0.0867	0.3805	0.8381	0.0275
FlowLens	0.1031	0.1066	0.0831	0.3055	0.1327	0.2872	0.7901	0.0263
Jaqen*	0.0935	0.0560	0.0422	0.3738	0.1704	2.5270	0.7832	0.0296
N3IC	0.0290	0.0372	0.0123	0.1120	0.0372	0.3346	0.9810	0.0278
NetBeacon	0.0478	0.0393	0.0163	0.4800	0.2974	0.2630	0.7486	0.0281
nPrintML	-0.0022	0.0043	-0.0027	0.0253	-0.0001	0.0002	0.9577	0.0253
Hypervision	0.0127	0.0034	0.0064	0.0971	0.0417	0.0336	0.5468	0.0158
FAE	0.3103	0.0110	0.0132	1.1539	0.7522	2.5234	0.2443	0.0201
FSC	0.3366	0.4045	0.1320	0.8474	0.3428	1.4277	0.7103	0.0178
Kitsune	0.3202	0.3720	0.2805	0.8810	0.3262	0.7243	0.6330	0.0511
Whisper	0.3078	0.0100	0.0126	0.4550	0.3273	2.1576	0.1805	0.0190
Overall	0.1467	0.1007	0.0573	0.4472	0.2285	0.9690	0.6740	0.0262

[1] ▲ indicates higher is better and ▼ indicates lower is better.
[2] PRC and ROC are short for AUPRC and AUROC.

(a) Reduced FPR with bounded TPR decrease.

(b) Improvements of AUROC and AUPRC.

(c) Improvements of accuracy and precision.

Figure 9: Accuracy improvements for various ML models.

The reason why pVoxel can accurately identify FPs and significantly improve the performance of various detection systems is that it analyzes the relationship between alarms (i.e., both FPs and FPs) which is represented by the topological features of points in the traffic feature space. Particularly, pVoxel utilizes voxels to represent massive points and analyzes density features to identify points in the high-density regions which denote FPs triggered by malicious flows. Meanwhile, it identifies FPs in the low-density regions that are triggered by diverse user behaviors. In addition, our FP identification method is unsupervised which allows pVoxel to reduce unforeseen FPs without requiring any prior knowledge, e.g., manually labeled FPs [11], original model parameters [25], and benign IP lists [80]. These goals cannot be achieved by existing methods [11, 24, 80].

(a) Num. of Trees (10-100).   (b) Depth of Trees (10-100).
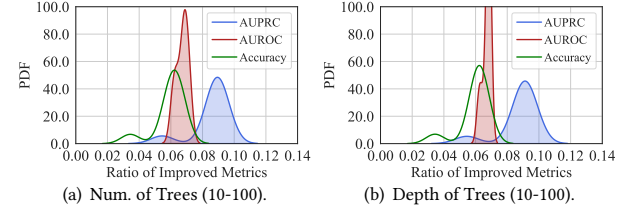
Figure 10: Accuracy improvements under various settings.

(a) CICFlowMeter*(Num.). (b) CICFlowMeter*(Depth).   (c) Whisper (K).

Figure 11: Reduced FPR and TPR under various settings.

(a) Naive Bayes.   (b) K-Means.   (c) SVM.

(d) Linear Classifier.   (e) Decision Tree.   (f) Logistic Classifier.

(g) KNN.   (h) DBSCAN.   (i) Autoencoder.

Figure 12: Comparing reduced FPR/TPR with retraining.

6.3 Robustness Evaluation

To evaluate the robustness of pVoxel, we establish benchmark detection methods by using nine ML models to classify the traffic in HyperVision datasets according to CICFlowMeter features [16]. These models cover a broad spectrum of existing models [31, 32, 59, 97], including both supervised and unsupervised models. The results show that pVoxel achieves robust FP reduction by identifying FPs raised by various ML models under different settings.

First, we validate that pVoxel is robust to various ML models. From Figure 9(a), the FPR reduced by pVoxel is ranging between $87.20\% \sim 99.05\%$. Meanwhile, the average TPR reduced by pVoxel is only 3.14%. In addition, Figure 9(b) and Figure 9(c) show that, by reducing FPR without significant TPR decrease, pVoxel improves
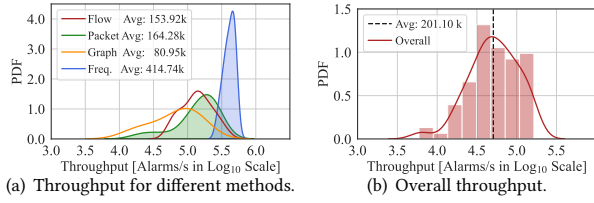
(a) Throughput for different methods. (b) Overall throughput.

Figure 13: Distribution of processing throughput.

(a) Flow and packet based methods. (b) Graph and frequency methods.
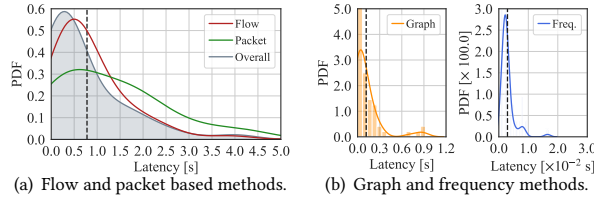
Figure 14: Distribution of latency for various methods.

(a) Latency and throughput. (b) Composition of Latency.
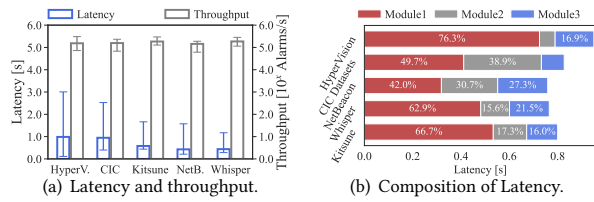
Figure 15: Performance of pVoxel on various datasets.

(a) Distribution of latency by modules. (b) Box plot of latency.
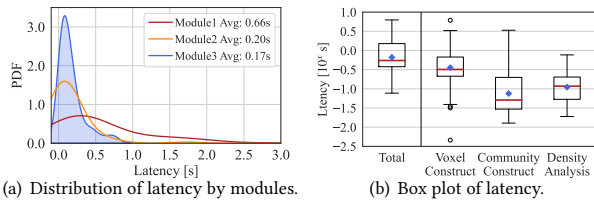
Figure 16: Latency incurred by different modules of pVoxel.

(a) Comparison of latency. (b) Comparison of throughput.
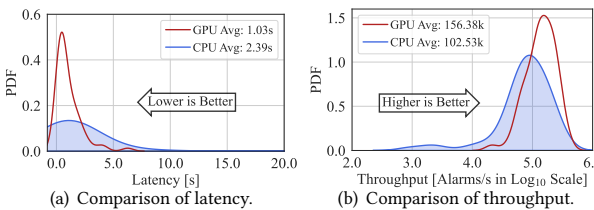
Figure 17: Comparison of performance on CPU and GPU.

AUPRC by 8.6%, AUROC by 30.86%, accuracy by 24.58%, and precision by 52.68%. Moreover, we observe that the accuracy improvements for unsupervised methods are higher, e.g., the AUROC improvement for the DBSCAN model is 64.37%. Since the FPR of unsupervised methods is normally higher than that of supervised ones, for instance, the FPR of the K-Means model is 3.33 times higher than SVM, and thus reducing such high FPR will significantly improve the performance of detection.

Second, we validate that pVoxel is robust to various settings. For this purpose, we adjust the hyper-parameters of the RF model and the K-Means model, given that these models are widely used [8, 11, 80, 97]. More precisely, we adjust the number of trees and the depth of trees for the RF model from 10 to 100 with a step length

of 10. Figure 11(a) and Figure 11(b) illustrate that pVoxel reduces 92.64% ∼ 97.99% FPR for different numbers of trees and 93.2% ∼ 98.28% FPR for different depths of trees with 0.24% and 0.33% TPR decrease, respectively. Similarly, from Figure 11(c), we find that pVoxel reduces 92.13% ∼ 96.19% FPR for different K-values of the K-Means model. Moreover, Figure 10 depicts the distributions of accuracy improvements when adjusting the hyper-parameters of the RF model. We can see that pVoxel improves AUPRC by 8.59%, AUROC by 6.71%, and accuracy by 5.94%, for different numbers of trees. Meanwhile, it improves AUPRC by 8.75%, AUROC by 6.72%, and accuracy by 5.96%, for different depths of trees. Thus, we conclude that pVoxel has high robustness when identifying FPs for various ML models with different settings.

The reason why pVoxel achieves robust FP identification is that pVoxel views different detection systems as black boxes (see Section 2.1 for details). Thus, it does not rely on any prior knowledge on ML models and their hyper-parameter settings. Instead, pVoxel identifies FPs according to the topological characteristics of the traffic feature space which is independent of specific ML models. Therefore, pVoxel is robust to various models with different settings.

6.4 Comparative Experiments

In this section, we compare pVoxel with traditional retraining methods when a portion of FPs are manually identified and included in training set to retrain models. First, we show that pVoxel outperforms retraining methods by large margins. Specifically, we randomly select 25%, 50%, and 75% FPs along with the equal ratios of TPs to retrain ML models and repeat the traffic detection using the original testing sets (samples used for retraining are excluded). Figure 12 compares the reduced FPR and TPR. Overall, pVoxel reduces 2.51 times higher FPR with 4.33 times lower TPR decrease than retraining methods. For the linear classifier (see Figure 12(d)), pVoxel reduces 93.19% FPR which is 1.85 times higher than the best performance of retraining. From Figure 12(e) and Figure 12(f), we can see that the retraining methods suffer from the catastrophic forgetting issue [25]. Particularly, the decision tree and the logistic classifier are unable to detect attack traffic after retraining and incur 28.24 and 15.45 times higher TPR decreases than pVoxel, respectively.

We compare pVoxel with the catastrophic forgetting mitigation method [25] that uses a regularization term to penalize a model if it cannot classify patterns due to large deviation of parameters. We use three λ values [25] as the weights of the term, and retrain the three state-of-the-art DNN based detection (see Table 5). Note that, the method [25] modifies DNN training algorithms and can only be applied to DNN based methods, while pVoxel can be applied to both DNN and non-DNN methods. We observe that pVoxel outperforms the methods and reduces 79.36% ∼ 84.64% higher FPR, which achieves similar TPR. Notably, when the weight is large (e.g., $\lambda = 5 \times 10^4$), the model will not update anymore so that retraining cannot reduce more FPs even if TPR does not significantly decrease. Besides, pVoxel achieves 38.98% ∼ 62.93% improvements over the mitigation method, when processing the FPs raised by the autoencoder based method (see Section 6.3).

The reason why pVoxel outperforms retraining methods is that pVoxel does not interfere with model training. By this way, it avoids

Table 5: Comparison against the forgetting mitigation.

Methods	Regularization Term [25]						pVoxel	
	$\lambda = 5 \times 10^2$		$\lambda = 5 \times 10^3$		$\lambda = 5 \times 10^4$			
Metrics	R.TPR	R.FPR	R.TPR	R.FPR	R.TPR	R.FPR	R.TPR	R.FPR
Kitsune	0.0428	0.3837	0.0419	0.2862	0.0004	0.2805	0.0342	0.9413
FAE	0.1129	0.1559	0.0337	0.1555	0.0245	0.1518	0.0526	0.9221
EULER	0.0000	0.0416	0.0000	0.0417	0.0000	0.0000	0.0475	0.9526
Average	0.0519	0.1937	0.0252	0.1611	0.0083	0.1441	0.0447	0.9386

the catastrophic forgetting issue [25] caused by model retraining. pVoxel views ML based detection methods as black-box systems and identifies FPs according to the topological relationship in the traffic feature space. Thus, it does not rely on any prior knowledge which is indispensable for model retraining, i.e., manually identified FPs, training datasets, and original parameters of ML models.

In addition, we observe that the traditional whitelist methods [11, 48] can only reduce few FPs. The reason is that, according to the existing study [80], most FPs are not triggered by the traffic generated from addresses owned by reputable organizations, which means that whitelists cannot effectively reduce FPs. Our empirical study also shows that lower than 0.1% FPs are triggered by the traffic of reputable ASes that own Alexa top 100 websites [5]. Besides, pVoxel does not require constructing whitelists, which further prevents exploiting whitelists, e.g., by IP spoofing [29, 55].

6.5 Throughput and Latency Evaluation

In this section, we show that pVoxel achieves high processing throughput with low latency when identifying FPs raised by various methods on different datasets. First, we validate that pVoxel achieves high processing throughput. Specifically, we measure the number of alarms processed by pVoxel within one second and plot the results in Figure 13(a). We observe that pVoxel achieves 153.92k, 164.28k, 80.95k and 414.74k alarms per second throughput for the methods that utilize flow-level, packet-level, graph-based, and the frequency features, respectively. The reason why pVoxel achieves higher throughput for the frequency feature based method is that pVoxel compresses the features to reduce the overhead. Figure 13(b) shows the distribution of throughput. pVoxel achieves over 201.10k alarms per second throughput.

Second, we show that pVoxel achieves high throughput with low latency. Figure 14(a) depicts the distribution of latency which illustrates that the average latency of pVoxel is 0.779s. Specifically, when identifying FPs raised by the methods using flow and packet based features, the average latency is only 0.946s and 1.310s, respectively. For graph and the frequency feature based methods (see Figure 14(b)), the average processing latency is significantly lower, i.e., 0.116s and 0.002s, because these methods raise fewer alarms. Moreover, from Figure 15(a), we observe that pVoxel has 0.986s bounded latency on different datasets, while achieving 144.40k ∼ 185.68k alarms per second high processing throughput.

Third, we examine the composition of the latency. We measure the latency incurred by the three modules of pVoxel individually and present the results in Figure 16. We can see that the three modules (i.e., voxel construction, community construction, and density analysis) incur 0.66s, 0.20s, and 0.17s latency, respectively. Similarly, Figure 15(b) shows the proportion of the latency incurred by the

modules. From Figure 16(b), we find that the first module consumes most time (64.07%) but it reduces the overhead of the other two modules, i.e., community construction (19.41%) and density analysis (16.50%), by aggregating massive points into voxels.

Finally, we compare the GPU implementation with multi-core CPU paralleled C++ implementation when analyzing the alarms raised by the seven flow feature based methods. Figure 17(a) compares the latency of the two implementations, where the GPU implementation reduces 56.90% latency of the CPU implementation. Also, from Figure 17(b), we observe that the throughput of the GPU implementation is 1.52 times higher than the CPU implementation. Because we offload the computation-intensive operations to GPU, i.e., voxel construction and the Floyd-Warshall algorithm for community construction.

The reason why pVoxel achieves high throughput and low latency is that the point cloud analysis leveraged by pVoxel is naturally able to analyze the topological features of massive points. In particular, pVoxel aggregates the massive points denoting many alarms into voxels, which significantly reduces the overhead. Moreover, pVoxel uses communities to represent adjacent voxels which further reduces the overhead. Eventually, the communities representing massive alarms allow pVoxel to perform the density analysis with low latency. Therefore, even if the traffic detection systems raise massive FPs, pVoxel can still classify each alarm as either TPs or FPs accurately and efficiently.

6.6 Evasion Attacks

Now we conduct experiments to show that attackers cannot easily exploit pVoxel to evade detection by simply mimicking benign users. Specifically, we validate the robustness against three evasion techniques according to recent studies [31, 33]: (i) Traffic obfuscation: injecting benign traffic into malicious flows; (ii) Adaptive traffic rates: attackers adjust packet rates to mimic benign flows; (iii) Manipulating flow features: attackers manipulate packet lengths to generate benign traffic patterns. Note that, these adaptive attacks can evade many existing methods [8, 59, 75, 97] that do not raise alarms under the evasions. Thereby, we collect raised alarms by three existing methods and apply pVoxel to analyze these alarms.

Table 6: Ratio of reduced TPs and FPs under evasion attacks.

Evasions	Obfuscation		Adaptive Rates		Faked Lengths		Overall	
Metrics	R.FPR	R.TPR	R.FPR	R.TPR	R.FPR	R.TPR	R.FPR	R.TPR
FAE	0.9613	0.0000	0.9275	0.0000	0.9623	0.0000	0.9503	0.0000
Whisper	0.9333	0.0052	0.9519	0.0057	0.9690	0.0156	0.9514	0.0088
HyperVision	0.9376	0.0073	0.9999	0.0033	0.9633	0.0059	0.9669	0.0055
Average	0.9440	0.0041	0.9597	0.0030	0.9648	0.0071	0.9561	0.0047

Table 6 shows that pVoxel reduces 94.40% ∼ 96.48% FPs under the evasion attacks. Meanwhile, the incurred TPR decrease is bounded by 1.56%. By reducing the majority of FPs without significant TPR decrease, pVoxel improves 15.65% ∼ 23.93% precision over the existing methods. Therefore, attackers cannot evade detection when pVoxel is applied. The reason is that features of the manipulated traffic generated by the evasion attacks are still in high-density regions. As a result, these high-density regions can still be effectively captured by pVoxel.

7 DISCUSSION

In this section, we discuss the practicality of pVoxel and its limitations.

7.1 Traffic Detection by Density Based Analysis

pVoxel utilizes density based analysis, which enables accurate and efficient FP detection. However, density based analysis cannot be directly applied to detect malicious traffic because it is difficult to analyze the complex distribution of all flow samples due to the high speed of traffic (e.g., 33.60M flows/h [87]). Our experiment results show that directly applying the analysis will incur low detection accuracy (i.e., AUC ≤ 0.75) and high latency (i.e., over 30 minutes). pVoxel well addresses these issues by only analyzing the distribution of positive samples (i.e., 0.10% ∼ 5.37% flows).

7.2 Integrating pVoxel into Training Pipelines

pVoxel can be integrated into training pipelines to facilitate model retraining. Specifically, it can feed FPs identified in testing phases into retraining processes, which can effectively correct misclassified traffic patterns. Thus, we can retrain the SVM, KNN, K-Means, DBSCAN, and the Bayes model (see Section 6.3). We observe that the decreased FPRs of the models range between 19.37% ∼ 51.37%. Thus, integrating pVoxel into training pipelines can effectively boost model performance.

Note that, the threat model in Section 2.2 considers detecting FPs under a realistic setting, i.e., pVoxel identifies FPs raised in testing phases without interfering with training pipelines. Since many commercial traffic detection systems are closed-source (e.g., Cisco ETA [19]), i.e., their training datasets and ML models are not publicly available, and thus FP reduction methods cannot be integrated into their training pipelines. pVoxel addresses this issue by reducing FPs at testing phases without requiring any prior knowledge in training phases, which achieves the effectiveness of FP reduction in practice.

7.3 Real-World Case Study

We randomly select 1,000 TPs and 1,000 FPs raised by CICFlowMeter* on the TLS vulnerability exploiting dataset [32]. CICFlowMeter* extracts 84 flow features [16] (e.g., the number of packets/bytes, the duration of a flow), and generates alarms when the feature vectors are classified as attack traffic. pVoxel classifies the alarms into TPs and FPs. We use t-SNE to map the high-dimensional features to 2D for visualization and then plot the points and the voxels denoting the alarms in Figure 18 and Figure 19, respectively.

The first module of pVoxel constructs 196 voxels to represent the 2,000 points denoting the alarms. To exclude isolated points, it identifies 38 voxels that represent sparsely distributed points, e.g., a voxel includes less than eight points. Finally, 1,809 retained points are represented by 158 voxels (see Figure 18(b) and Figure 19(a)). Meanwhile, the 191 isolated points are excluded, which consist of 3 TPs and 187 FPs. Note that, the 3 TPs are misclassified because the associated flow duration features are obviously higher. The three points drift from the points denoting other attack flows and thus are labeled as isolated points of FPs. We speculate that these flows are constructed to check the reachability of victim websites, thereby differing from the actual vulnerability exploiting flows.
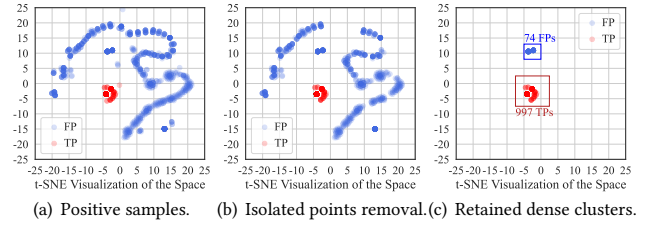

(a) Positive samples. (b) Isolated points removal. (c) Retained dense clusters.

Figure 18: Point distribution analyzed by each module.

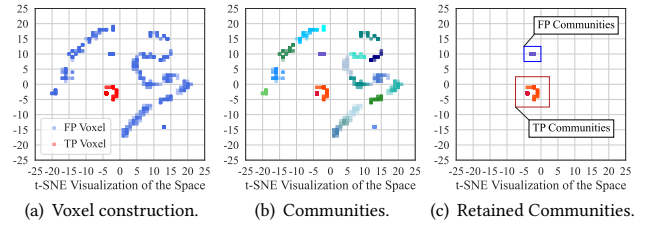(a) Voxel construction. (b) Communities. (c) Retained Communities.

Figure 19: Voxel distribution analyzed by each module.

The second module merges the voxels into 19 communities that are depicted in Figure 19(b), and the last module identifies 16 communities denoting sparse clusters of FPs with low densities, i.e., 22 ∼ 57 points per community. pVoxel labels the other three communities as TPs that are associated with significantly higher densities, e.g., 908 points per community (see Figure 19(c)). Note that, a small number of FPs are not removed. These FPs are triggered by repetitive DNS queries from a user, which deviate from normal user behaviors (see Figure 18(c)). Finally, pVoxel reduces 92.60% FPR with 0.30% negligible TPR decrease.

8 RELATED WORK

ML Based Malicious Traffic Detection. ML based detection achieves higher accuracy than traditional signature based detection [12]. In particular, they can detect unknown attacks [31, 59]. For generic detection, Fu *et al.* leveraged the frequency domain features to achieve accurate detection [31]. Barradas *et al.* developed FlowLens to extract distribution features and detected attacks by using Random Forest [8]. Similarly, Zhou *et al.* [97] developed NetBeacon to install ML models on programmable switches. Panda *et al.* [65] built flow based detection upon SmartNICs. Mirsky *et al.* developed Kitsune to learn per-packet features by using autoencoders [59]. Moreover, HyperVision utilized graphs to detect encrypted malicious traffic [32]. For task specific detection, Nelms *et al.* [61], Invernizzi *et al.* [40], and Bilge *et al.* [11] detected different behaviors of malware campaigns. Bartos *et al.* [9], Tang *et al.* [79], and Dodia *et al.* [23] detected malformed Web traffic. Wichtlhuber *et al.* [86] and Wagner *et al.* [84] throttled DDoS traffic at IXP. Dodia *et al.* [81] and Sharma *et al.* [71] captured attack traffic targeting IoT devices. Note that, malicious traffic detection is different from traffic classification which classifies whether traffic is generated by a certain application [6, 70, 74, 82].

Reducing FPs for ML Based Malicious Traffic Detection. Most existing studies do not perform FP reduction [8, 32, 39, 97] even if the FPR is unacceptable [4, 76]. In general, existing studies mitigated the high FPR issue by using whitelists and model retraining. For whitelist based methods, Tegeler *et al.* [80] reduced FPR of malware

traffic detection by excluding FPs triggered by IP addresses held by trusted organizations. Similarly, Bilge *et al.* [11] reduced the issue by constructing IP whitelists according to manually identified FPs and the reputation of ASes. Note that, attackers can utilize IP whitelists to evade detection via IP spoofing [55]. For retraining based methods, Du *et al.* [24] incrementally retrained the DNN models for traffic log based detection using FPs reported by user feedback and mitigated the catastrophic forgetting issues [25]. These methods merely reduce FPs according to manually identified FPs, which are different from our automatic FPs identification. Similarly, other ML based security applications also suffer from high FPRs. They also utilized model retraining [7, 15, 38, 45, 52, 90] or prior knowledge [2, 42, 51, 73, 91, 93, 98] to reduce FPs.

Analyzing ML Based Security Applications. Arp *et al.* [4] studied the practical issues of ML based systems for security, showing the high FPR issues of these systems [24, 59]. Recently, Alahmadi *et al.* [1] showed that deploying ML based systems in SOCs is hindered by the high FPR issue. Moreover, Sommer *et al.* [76] analyzed the reason why ML based anomaly detection suffers from low usability. Moreover, Han *et al.* [37], Jacobs *et al.* [41], and Wei *et al.* [85] addressed the explainability issue of traffic detection, which is orthogonal to pVoxel.

9 CONCLUSION

In this paper, we propose pVoxel, which identifies FPs for ML based traffic detection systems. Specifically, pVoxel treats the traffic feature vector associated with each alarm as a point in the traffic feature space. It utilizes point cloud analysis to capture the topological features among the points and classifies the alarms into TPs and FPs. In particular, we aggregate the points into voxels, i.e, cubes in the high-dimensional traffic feature space, and develop an unsupervised method to identify the voxels denoting FPs according to their density features. Our experiments with 75 real-world attacks demonstrate that pVoxel outperforms retraining based methods by reducing 95.52% FPs for 11 state-of-the-art traffic detection methods under various settings. Meanwhile, pVoxel achieves efficient alarm processing and can process 201.10 thousand alarms per second.

REFERENCES

[1] Bushra A. Alahmadi et al. 2022. 99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms. In *Security*. USENIX, 2783–2800.
[2] Abdulellah Alsaheel et al. 2021. ATLAS: A Sequence-based Learning Approach for Attack Investigation. In *Security*. USENIX, 3005–3022.
[3] ArcSight. Accessed Sep. 2023. Enterprise Security Manager. https://www.microfocus.com/en-us/cyberres/secops/arcsight-esm.
[4] Daniel Arp et al. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *Security*. USENIX.
[5] AWS. accessed Sep. 2023. Alexa Top Websites. https://www.expireddomains.net/alexa-top-websites/.
[6] Alireza Bahramali et al. 2020. Practical Traffic Analysis Attacks on Secure Messaging Applications. In *NDSS*. ISOC.
[7] Federico Barbero et al. 2022. Transcending TRANSCEND: Revisiting Malware Classification in the Presence of Concept Drift. In *SP*. IEEE, 805–823.
[8] Diogo Barradas et al. 2021. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications. In *NDSS*. ISOC.
[9] Karel Bartos et al. 2016. Optimized Invariant Representation of Network Traffic for Detecting Unseen Malware Variants. In *Security*. USENIX, 807–822.
[10] Hugo L. J. Bijmans et al. 2019. Just the Tip of the Iceberg: Internet-Scale Exploitation of Routers for Cryptojacking. In *CCS*. ACM, 449–464.
[11] Leyla Bilge et al. 2012. Disclosure: detecting botnet command and control servers through large-scale NetFlow analysis. In *ACSAC*. ACM, 129–138.
[12] Anna L. Buczak and Erhan Guven. 2016. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Commun. Surv.*
Tutorials 18, 2 (2016), 1153–1176.
[13] Yue Cao et al. 2016. Off-Path TCP Exploits: Global Rate Limit Considered Dangerous. In *Security*. USENIX, 209–225.
[14] Evan Caville et al. 2022. Anomal-E: A self-supervised network intrusion detection system based on graph neural networks. *Knowl. Based Syst.* 258 (2022), 110030.
[15] Yizheng Chen et al. 2020. On Training Robust PDF Malware Classifiers. In *Security*. USENIX, 2343–2360.
[16] CIC. Accessed Sep. 2023. CICFlowMeter: a network traffic flow generator and analyser. https://www.unb.ca/cic/research/applications.html.
[17] CIC. Accessed Sep. 2023. DDoS Evaluation Datasets (CIC-DDoS2019). https://www.unb.ca/cic/datasets/ddos-2019.html.
[18] CIC. Accessed Sep. 2023. Intrusion Detection Evaluation Datasets (CIC-IDS2017). https://www.unb.ca/cic/datasets/ids-2017.html.
[19] Cisco. Accessed Sep. 2023. Cisco Encrypted Traffic Analytics. https://www.cisco.com/c/en/us/solutionsenterprise-networks/enterprise-network-security/eta.html.
[20] Cisco. Accessed Sep. 2023. Cisco SPAN. https://www.cisco.com/c/en/us/support/docs/swit-ches/catalyst-6500-series-switches/10570-41.html.
[21] David Coupier. 2019. *Stochastic Geometry Modern Research Frontiers*. Vol. 1. Springer.
[22] Peng Dai et al. 2020. Neural Point Cloud Rendering via Multi-Plane Projection. In *CVPR*. CVF/IEEE, 7827–7836.
[23] Priyanka Dodia et al. 2022. Exposing the Rat in the Tunnel: Using Traffic Analysis for Tor-based Malware Detection. In *CCS*. ACM, 875–889.
[24] Min Du et al. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *CCS*. ACM, 1285–1298.
[25] Min Du et al. 2019. Lifelong Anomaly Detection Through Unlearning. In *CCS*. ACM, 1283–1297.
[26] Zakir Durumeric et al. 2013. ZMap: Fast Internet-wide Scanning and Its Security Applications. In *Security*. USENIX, 605–620.
[27] Zakir Durumeric et al. 2014. An Internet-Wide View of Internet-Wide Scanning. In *Security*. USENIX, 65–78.
[28] Cristian Estan and George Varghese. 2003. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.* 21, 3 (2003), 270–313.
[29] Xuewei Feng et al. 2020. Off-Path TCP Exploits of the Mixed IPID Assignment. In *CCS*. ACM, 1323–1335.
[30] Xuewei Feng et al. 2022. Off-Path Network Traffic Manipulation via Revitalized ICMP Redirect Attacks. In *Security*. USENIX, 2619–2636.
[31] Chuanpu Fu et al. 2021. Realtime Robust Malicious Traffic Detection via Frequency Domain Analysis. In *CCS*. ACM, 3431–3446.
[32] Chuanpu Fu et al. 2023. Detecting Unknown Encrypted Malicious Traffic in Real Time via Flow Interaction Graph Analysis. In *NDSS*. ISOC.
[33] Chuanpu Fu et al. 2023. Frequency Domain Feature Based Robust Malicious Traffic Detection. *IEEE/ACM Trans. Netw.* 31, 1 (2023), 452–467.
[34] Enrico Gobbetti and Fabio Marton. 2005. Far voxels: a multiresolution framework for interactive rendering of huge complex 3D models on commodity graphics platforms. *ACM Trans. Graph.* 24, 3 (2005), 878–885.
[35] Harm Griffioen et al. 2021. Scan, Test, Execute: Adversarial Tactics in Amplification DDoS Attacks. In *CCS*. ACM, 940–954.
[36] Harm Griffioen and Christian Doerr. 2020. Examining Mirai's Battle over the Internet of Things. In *CCS*. ACM, 743–756.
[37] Dongqi Han et al. 2021. DeepAID: Interpreting and Improving Deep Learning-based Anomaly Detection in Security Applications. In *CCS*. ACM, 3197–3217.
[38] Xueyuan Han et al. 2020. Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *NDSS*. ISOC.
[39] Jordan Holland et al. 2021. New Directions in Automated Traffic Analysis. In *CCS*. ACM, 3366–3383.
[40] Luca Invernizzi et al. 2014. Nazca: Detecting Malware Distribution in Large-Scale Networks. In *NDSS*. ISOC.
[41] Arthur Selle Jacobs et al. 2022. AI/ML for Network Security: The Emperor has no Clothes. In *CCS*. ACM, 1537–1551.
[42] Steve T. K. Jan et al. 2020. Throwing Darts in the Dark? Detecting Bots with Limited Data using Neural Data Augmentation. In *SP*. IEEE, 1190–1206.
[43] Mobin Javed and Vern Paxson. 2013. Detecting stealthy, distributed SSH brute-forcing. In *CCS*. ACM, 85–96.
[44] Mattijs Jonker et al. 2017. Millions of targets under attack: a macroscopic characterization of the DoS ecosystem. In *IMC*. ACM, 100–113.
[45] Roberto Jordaney et al. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *Security*. USENIX, 625–642.
[46] Min Suk Kang et al. 2013. The Crossfire Attack. In *SP*. IEEE, 127–141.
[47] Isaiah J. King and H. Howie Huang. 2022. Euler: Detecting Network Lateral Movement via Scalable Temporal Graph Link Prediction. In *NDSS*. ISOC.
[48] Faris Bugra Kokulu et al. 2019. Matched and Mismatched SOCs: A Qualitative Study on Security Operations Center Issues. In *CCS*. ACM, 1955–1970.
[49] Daniel Kopp et al. 2019. DDoS Hide & Seek: On the Effectiveness of a Booter Services Takedown. In *IMC*. ACM, 65–72.

[50] Aleksandar Kuzmanovic and Edward W. Knightly. 2003. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *SIGCOMM*. ACM, 75–86.

[51] Zhou Li et al. 2013. Finding the Linchpins of the Dark Web: a Study on Topologically Dedicated Hosts on Malicious Web Infrastructures. In *SP*. IEEE, 112–126.

[52] Fucheng Liu et al. 2019. Log2vec: A Heterogeneous Graph Embedding Based Approach for Detecting Cyber Threats within Enterprise. In *CCS*. ACM, 1777–1794.

[53] Lingjie Liu et al. 2020. Neural Sparse Voxel Fields. In *NIPS*. Curran.

[54] Zaoxing Liu et al. 2021. Jaqen: A High-Performance Switch-Native Approach for Detecting and Mitigating Volumetric DDoS Attacks with Programmable Switches. In *Security*. USENIX, 3829–3846.

[55] Matthew J. Luckie et al. 2019. Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet. In *CCS*. ACM, 465–480.

[56] Xiapu Luo and Rocky K. C. Chang. 2005. On a New Class of Pulsing Denial-of-Service Attacks and the Defense. In *NDSS*. ISOC.

[57] Robert Merget et al. 2019. Scalable Scanning and Automatic Classification of TLS Padding Oracle Vulnerabilities. In *Security*. USENIX, 1029–1046.

[58] Sadegh Momeni Milajerdi et al. 2019. HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows. In *SP*. IEEE, 1137–1152.

[59] Yisroel Mirsky et al. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In *NDSS*. ISOC.

[60] mlpack. accessed Sep. 2023. An open Source Machine Learning Library. https://www.mlpack.org/.

[61] Terry Nelms et al. 2015. WebWitness: Investigating, Categorizing, and Mitigating Malware Download Paths. In *Security*. USENIX, 1025–1040.

[62] Arman Noroozian et al. 2019. Platforms in Everything: Analyzing Ground-Truth Data on the Anatomy and Economics of Bullet-Proof Hosting. In *Security*. USENIX, 1341–1356.

[63] NVIDIA. Accessed Sep. 2023. CUDA: a parallel computing platform on GPU. https://developer.nvidia.com/cuda-toolkit.

[64] Adam Oest et al. 2020. Sunrise to Sunset: Analyzing the End-to-end Life Cycle and Effectiveness of Phishing Attacks at Scale. In *Security*. USENIX, 2039–2056.

[65] Sourav Panda et al. 2021. SmartWatch: accurate traffic analysis and flow-state tracking for intrusion prevention using SmartNICs. In *CoNEXT*. ACM, 60–75.

[66] pVoxel. Accessed Sep. 2023. Code realse. https://github.com/fuchuanpu/pVoxel.

[67] Charles Ruizhongtai Qi et al. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NIPS*. Curran Associates, 5099–5108.

[68] Sivaramakrishnan Ramanathan et al. 2020. BLAG: Improving the Accuracy of Blacklists. In *NDSS*. ISOC.

[69] Philipp Richter and Arthur W. Berger. 2019. Scanning the Scanners: Sensing the Internet from a Massively Distributed Network Telescope. In *IMC*. ACM, 144–157.

[70] Vera Rimmer et al. 2018. Automated Website Fingerprinting through Deep Learning. In *NDSS*. ISOC.

[71] Rahul Anand Sharma et al. 2022. Lumen: a framework for developing and evaluating ML-based IoT network anomaly detection. In *CoNEXT*. ACM, 59–71.

[72] Meng Shen et al. 2021. Accurate Decentralized Application Identification via Encrypted Traffic Analysis Using Graph Neural Networks. *IEEE Trans. Inf. Forensics Secur.* 16 (2021), 2367–2380.

[73] Yun Shen et al. 2022. A Large-scale Temporal Measurement of Android Malicious Apps: Persistence, Migration, and Lessons Learned. In *Security*. USENIX, 1167–1184.

[74] Sandra Siby et al. 2020. Encrypted DNS -> Privacy? A Traffic Analysis Perspective. In *NDSS*. ISOC.

[75] Giuseppe Siracusano et al. 2022. Re-architecting Traffic Analysis with Neural Network Interface Cards. In *NSDI*. USENIX, 513–533.

[76] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *SP*. IEEE, 305–316.

[77] Dietrich Stoyan et al. 2013. *Stochastic Geometry and Its Applications*. Vol. 1. WILEY.

[78] Suricata. Accessed Sep. 2023. An Open Source Threat Detection Engine. https://suricata-ids.org/.

[79] Ruming Tang et al. 2020. ZeroWall: Detecting Zero-Day Web Attacks through Encoder-Decoder Recurrent Neural Networks. In *INFOCOM*. IEEE, 2479–2488.

[80] Florian Tegeler et al. 2012. BotFinder: finding bots in network traffic without deep packet inspection. In *CoNEXT*. ACM, 349–360.

[81] Ege Tekiner et al. 2022. A Lightweight IoT Cryptojacking Detection Mechanism in Heterogeneous Smart Home Networks. In *NDSS*. ISOC.

[82] Thijs van Ede et al. 2020. FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic. In *NDSS*. ISOC.

[83] Thijs van Ede et al. 2022. DEEPCASE: Semi-Supervised Contextual Analysis of Security Events. In *SP*. IEEE, 522–539.

[84] Daniel Wagner et al. 2021. United We Stand: Collaborative Detection and Mitigation of Amplification DDoS Attacks at Scale. In *CCS*. ACM, 970–987.

[85] Feng Wei et al. 2023. XNIDS: Explaining Deep Learning-based Network Intrusion Detection Systems for Active Intrusion Responses. In *Security*. USENIX, to appear.

[86] Matthias Wichtlhuber et al. 2022. IXP scrubber: learning from blackholing traffic for ML-driven DDoS detection at scale. In *SIGCOMM*. ACM, 707–722.

[87] WIDE. Accessed Sep. 2023. MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp/mawi/.

[88] Jiarong Xing et al. 2020. NetWarden: Mitigating Network Covert Channels while Preserving Performance. In *Security*. USENIX, 2039–2056.

[89] Jiarong Xing et al. 2021. Ripple: A Programmable, Decentralized Link-Flooding Defense Against Adaptive Adversaries. In *Security*. USENIX, 3865–3880.

[90] Limin Yang et al. 2021. CADE: Detecting and Explaining Concept Drift Samples for Security Applications. In *Security*. USENIX, 2327–2344.

[91] Runqing Yang et al. 2020. UIScope: Accurate, Instrumentation-free, and Visible Attack Investigation for GUI Applications. In *NDSS*. ISOC.

[92] Zeek. Accessed Sep. 2023. An Open Source Network Security Monitoring Tool. https://zeek.org/.

[93] Jun Zeng et al. 2021. WATSON: Abstracting Behaviors from Audit Logs via Aggregation of Contextual Semantics. In *NDSS*. ISOC.

[94] Dimitris Zermas et al. 2017. Fast segmentation of 3D point clouds: A paradigm on LiDAR data for autonomous vehicle applications. In *ICRA*. IEEE, 5067–5073.

[95] Menghao Zhang et al. 2020. Poseidon: Mitigating Volumetric DDoS Attacks with Programmable Switches. In *NDSS*. ISOC.

[96] Jing Zheng et al. 2018. Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis. *IEEE Trans. Inf. Forensics Secur.* 13, 7 (2018), 1838–1853.

[97] Guangmeng Zhou et al. 2023. NetBeacon: An Efficient Design of Intelligent Network Data Plane. In *Security*. USENIX, to appear.

[98] Shuofei Zhu et al. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *Security*. USENIX, 2361–2378.

[99] Shitong Zhu et al. 2020. You do (not) belong here: detecting DPI evasion attacks with context learning. In *CoNEXT*. ACM, 183–197.

A DETAILS OF THEORETICAL ANALYSIS

A.1 Proof of Theorem 5.3 and Theorem 5.4

PROOF. *Theorem 5.3, the ratio of the densities for accumulative flow features.*

First, we analyze the expected density of points denoting FPs and TPs for accumulative flow features, i.e., $f_E(\cdot) = \text{Sum}(\cdot)$. Given that each random variable in $\vec{s}_i = [s_{i1}, \ldots, s_{iL}]^T$ ($1 \leq i \leq N$) obeys Gaussian distribution, i.e, $s_{ij} \sim \mathcal{N}(0, \sigma^2)$ holds for $\forall\, 1 \leq j \leq L$, we can obtain the distribution of the accumulative flow features V_i for the ith flow:

$$V_i = \sum_{j=1}^{L} s_{ij} \sim \mathcal{N}(0, L\sigma^2).$$

According to the definition of the range R (see Definition 5.1), we derive $E[R_{\text{Sum}}]$ by using the order statistic equation. That is, the cumulative distribution function of the rth order statistic $X_{(r)}$ over N iid. variables $[X_1, \ldots, X_N]^T$ equals to:

$$F_{X_{(r)}}(x) = \sum_{j=r}^{N} \binom{N}{j} [F_X(x)]^j * [1 - F_X(x)]^{N-j}.$$

Moreover, we use $\varphi_v(x)$ and $\Phi_v(x)$ to indicate the cumulative distribution function (CDF) and probability density function (PDF) of each element in $\vec{V} = [V_1, \ldots, V_N]$, respectively. Then, we can obtain the following results:

$$E[R_{\text{Sum}}] = E[\max(\vec{V})] - E[\min(\vec{V})]$$

$$= \int_{-\infty}^{+\infty} x \cdot N \cdot \varphi_v(x) \left([\Phi_v(x)]^{N-1} - [1 - \Phi_v(x)]^{N-1} \right) dx$$

$$= \sqrt{\frac{2}{\pi}} \cdot N \cdot \sigma \cdot \sqrt{L} \cdot 2^{1-N} \cdot$$

$$\int_{-\infty}^{+\infty} u e^{-u^2} \left([1 + \text{erf}(u)]^{N-1} - [1 - \text{erf}(u)]^{N-1} \right) du,$$

where $u = \dfrac{x}{\sqrt{2L}\sigma}$, and $\text{erf}(z) = \dfrac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$.

According to the relationship between the density and the range (Definition 5.2), we derive the ratio of the densities associated with benign flows (FPs) and malicious flows (TPs):

$$\frac{\text{E}[D_\text{B}]}{\text{E}[D_\text{M}]} = \frac{\text{E}[R_\text{Sum}|\sigma_\text{M}]}{\text{E}[R_\text{Sum}|\sigma_\text{B}]} = \left(\frac{\sigma_\text{B}}{\sigma_\text{M}}\right)^{-1} \left(\frac{\sqrt{L_\text{B}}}{\sqrt{L_\text{M}}}\right)^{-1}.$$

Here, we complete the proof for Theorem 5.3, which shows that the density of points denoting TPs is higher than that denoting FPs, since the variance of attack flows σ_M and the length of the flows L_M are significantly smaller according to empirical studies [28, 32, 69]. It also means that pVoxel can effectively identify FPs by detecting the points in the low-density regions of the feature space. □

Proof. *Theorem 5.4, the ratio of the densities for average based flow features.*

The proof for Theorem 5.4 is similar to the proof for Theorem 5.3. When $f_\text{E}(\cdot) = \text{Avg}(\cdot)$, the average based flow features denoted by V_i equals to:

$$V_i = \frac{1}{L} \sum_{j=1}^L s_{ij} \sim \mathcal{N}(0, \frac{\sigma^2}{L}).$$

Similarly, we can obtain the expected range for the flow features R_Avg according to the equation of order statistics:

$$\text{E}[R_\text{Avg.}] = \sqrt{\frac{2}{\pi}} \cdot \frac{N}{2^{(N-1)}} \cdot \frac{\sigma}{\sqrt{L}} \cdot \psi(N),$$

where $\psi(N)$ is a transcendental integral function, and $\text{erfc}(z) = 1 - \text{erf}(z)$:

$$\psi(N) = \int_{-\infty}^{+\infty} u e^{-u^2} \left([1 + \text{erf}(u)]^{N-1} - [\text{erfc}(u)]^{N-1}\right) du,$$

According to Definition 5.2 and the inequalities of the parameters, i.e., $\sigma_\text{B} > \sigma_\text{M}$ and $\text{E}[L_\text{B}] > \text{E}[L_\text{M}]$, we derive the ratio of the densities of points representing FPs and TPs triggered by benign and malicious flows:

$$\frac{\text{E}[D_\text{B}]}{\text{E}[D_\text{M}]} = \frac{\text{E}[R_\text{Avg.}|\sigma_\text{M}]}{\text{E}[R_\text{Avg.}|\sigma_\text{B}]} = \left(\frac{\sigma_\text{B}}{\sigma_\text{M}}\right)^{-1} > 1.$$

Therefore, the average based flow features (e.g., the average packet size of a flow) of benign flows exhibit sparse distribution in the feature space. pVoxel captures these features as isolated points and sparse clusters in the voxel construction module (Section 4.1) and the density analysis module (Section 4.3), respectively. □

A.2 Proof of Theorem 5.5 and Theorem 5.6

Proof. *Theorem 5.5, the upper bound of the expected densities for range based flow features.*

When $f_\text{E}(\cdot) = \text{Range}(\cdot)$, the range based features can be represented by $V_i = V_i^{(r)} - V_i^{(l)}$, where $V_i^{(r)} = \max(\vec{s}_i)$ and $V_i^{(l)} = \min(\vec{s}_i)$. We split the range R_Range according to the original point, i.e., $R_\text{Range} = R^{(l)} + R^{(r)}$, where $R^{(l)}$ and $R^{(r)}$ denote the lengths of the left and the right parts, respectively. After that, we examine the left part of R_Range:

$$\text{E}[R^{(l)}] = -\text{E}[\min(V_1^{(l)}, \ldots, V_N^{(l)})]$$
$$= -\text{E}[\min(s_{11}, \ldots, s_{ij}, \ldots, s_{NL})] \quad (1 \le i \le N, 1 \le j \le L)$$

We use $f_{R^{(l)}}(x)$ to indicate the PDF of $R^{(l)}$. According to the order statistic equation, we formulate $f_{R^{(l)}}(x)$ as:

$$f_{R^{(l)}}(x) = NL \cdot \varphi_s(x) \cdot [1 - \Phi_s(x)]^{NL-1},$$

where $\varphi_s(x)$ and $\Phi_s(x)$ are the PDF and CDF of the variables in $\vec{s}_i$ with the standard variance of σ. In the next step, we derive the expectation of $R^{(l)}$:

$$\text{E}[R^{(l)}] = -\int_{-\infty}^{+\infty} x \cdot f_{R^{(l)}}(x) dx$$
$$= -\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} \cdot \frac{Kx}{2^{K-1}} \cdot [1 - \text{erf}(\frac{x}{\sqrt{2}\sigma})]^{K-1} dx,$$

where $K = NL$. Let $u = x/\sqrt{2}\sigma$, and $dx = \sqrt{2}\sigma du$:

$$\text{E}[R^{(l)}] = -\sqrt{\frac{2}{\pi}} \cdot K\sigma \cdot \int_{-\infty}^{+\infty} h(u) \cdot g(u) du.$$

$$\begin{cases} h(u) &= u e^{-u^2} \\ g(u) &= \left[\frac{1 - \text{erf}(u)}{2}\right]^{K-1} \end{cases}$$

Here, we investigate the monotonicity of $g(u)$:

$$g'(u) = -\frac{K}{2^{K-1}\sqrt{\pi}} \cdot e^{-x^2} \cdot [1 - \text{erf}(u)]^{K-1} \ge 0,$$

$$g''(u) = 2^{-K}\left[\frac{1}{\pi} \cdot 4e^{-2x^2} \cdot (K-1)K \cdot (1 - \text{erf}(u))^{K-2} + \right.$$
$$\left. \frac{1}{\sqrt{\pi}} \cdot 4e^{-x^2} \cdot Kx \cdot (1 - \text{erf}(u))^{K-1}\right].$$

When $g''(u_0) = 0$, we can observe that:

$$\lim_{K \to +\infty} g'(u_0) = -\infty, \quad \lim_{K \to +\infty} g(u_0) = \frac{1}{2}, \quad \lim_{K \to +\infty} 1 - \text{erf}(u_0) = 2.$$

Since $K = LN$ is a large value, we can obtain an approximate value for u_0:

$$u_0 = -\sqrt{\frac{W_0\left(\frac{K^2-1}{2\pi}\right)}{2}},$$

where W_0 is the first Lambert W function. Moreover, it is obvious that:

$$\lim_{K \to +\infty, u \to u_0^+} g(u) = 1, \quad \lim_{K \to +\infty, u \to u_0^-} g(u) = 0.$$

Thus, we can use a step function $\tilde{g}(u)$ to approach $g(u)$, and obtain a lower bound of the expected value of $R^{(l)}$ by discarding the integral over $x > u_0$:

$$\tilde{g}(u) = \begin{cases} 1, & \text{if } x \le u_0, \\ 0, & \text{else}, \end{cases}$$

$$\text{E}[R^{(l)}] \ge \sqrt{\frac{2}{\pi}} \cdot K\sigma \int_{-\infty}^{+\infty} h(u) \cdot \tilde{g}(u) du = \sqrt{\frac{2}{\pi}} \cdot K\sigma \int_{-\infty}^{u_0} u e^{-u} du$$
$$= \sqrt{\frac{2}{\pi}} \cdot K\sigma \left[\frac{e^{-u^2}}{2}\Big|_{-\infty}^{u_0}\right] = \frac{\sigma NL}{NL-1} \sqrt{W_0\left(\frac{N^2L^2-1}{2\pi}\right)},$$

which is a lower bound of $\text{E}[R^{(l)}]$ with an error of $o(0.123)$.

In the following steps, we prove that the lower bound is increasing over the range of K:

$$\text{E}[R^{(l)}] \ge \sigma \vartheta(K), \quad \vartheta(K) = \frac{K}{K-1} \sqrt{W_0\left(\frac{K^2-1}{2\pi}\right)}.$$

To achieve this, we derive the first-order derivative of $\vartheta(K)$:

$$\vartheta'(K) = \frac{\sqrt{\tilde{W}(K)}}{K-1} - \frac{K\sqrt{\tilde{W}(K)}}{(K-1)^2} + \frac{K\sqrt{\tilde{W}(K)}}{(K-1)(K^2-1)(1+\sqrt{\tilde{W}(K)})},$$

where $\tilde{W}(K) = W_0(\frac{K^2-1}{2\pi})$. Note that, $\vartheta'(K) \geq 0$ holds when $K^2 \geq (1+K)(1+\tilde{W}(K))$. Given that K is large, the condition is equivalent to $K \geq (1+\tilde{W}(K))$. According to the inequality of the Lambert W function:

$$W_0(K) \leq \ln\left(\frac{2K+1}{1+\ln(K+1)}\right),$$

we prove that the lower bound is monotonically increasing:

$$K \geq 1 + \ln(2K+1) \geq 1 + \ln\left(\frac{2K+1}{1+\ln(K+1)}\right) \geq 1 + \tilde{W}(K)$$

$$\Rightarrow \vartheta'(K) \geq 0.$$

Moreover, it is obvious that $E[R^{(r)}] = E[R^{(l)}]$ according to the symmetry of the space:

$$E[R_{\text{Range}}] \geq \frac{2\sigma NL}{NL-1}\sqrt{W_0\left(\frac{N^2L^2-1}{2\pi}\right)}$$

Finally, we obtain the upper bound for $E[D_{\text{Range}}]$ according to Definition 5.2. The result shows that the upper bound of the density is inversely proportional to σ and monotonically decreasing over the whole range of L. □

PROOF. *Theorem 5.6, the upper bound of the expected densities for minimum value based flow features.*

Now, we consider the minimum value based features, i.e., $f_E(\cdot) = \text{Min}(\cdot)$ and $V_i = \min(\vec{s}_i)$. Following the proof for Theorem 5.5, we split the range into $R^{(l)}$ and $R^{(r)}$. It is clear that $E[R^{(l)}]$ of minimum value based features equals to that of range based features. Moreover, we calculate an upper bound for $E[R^{(r)}]$ according to Jensen's inequality:

$$E[R^{(r)}] = -E[\max(V_1, \ldots, V_N)] \leq -\max(E[V_1], \ldots, E[V_N])$$

$$\leq -E[V_i] = E[\min(\vec{s}_i)] \quad (1 \leq i \leq N).$$

We use $f_{V_i}(x)$ to indicate the PDF of V_i and derive the upper bound for the right part of the range:

$$f_{V_i}(x) = -L \cdot \varphi_s(x)[1 - \Phi_s(x)]^{L-1}$$

$$E[R^{(r)}] \leq -\int_{-\infty}^{+\infty} x \cdot f_{V_i}(x)dx$$

$$= -\sqrt{\frac{2}{\pi}} \cdot L\sigma \cdot \int_{-\infty}^{+\infty} ue^{-u^2} \cdot \left[\frac{\text{erfc}(u)}{2}\right]^{L-1} du$$

After that, we use the same way as the proof for Theorem 5.5 to approach the bound:

$$E[R^{(r)}] \leq \frac{2\sigma L}{L-1}\sqrt{W_0\left(\frac{L^2-1}{2\pi}\right)} + o(0.123)\sigma$$

Combined with the lower bound derived before and the Definition 5.2, we can obtain the upper bound of the density for the minimum value based features. □

A.3 Proof of Theorem 5.7 and Theorem 5.8

In this section, we prove the theorems for variance based flow features. Particularly, we consider the cases for large and small L separately. When $f_E(\cdot) = \text{Var}(\cdot)$, the extracted variance based flow features for i^{th} flow is:

$$V_i = \frac{\sum_{j=1}^{L} s_{ij}^2}{L} \sim \frac{\sigma^2}{L}\chi^2(L),$$

where $\chi^2(L)$ is a variable that obeys Chi-squared distribution with L degrees of freedom. According to the equation of order statistics, we formulate $E[R_{\text{Var}}]$ as:

$$E[R_{\text{Var}}] = \frac{\sigma^2}{L} \cdot \int_0^{+\infty} Nx \cdot f_{\chi^2(L)}(x) \cdot [F_{\chi^2(L)}(x)]^{N-1}dx,$$

where $f_{\chi^2(L)}(x)$ and $F_{\chi^2(L)}(x)$ are the PDF and CDF of the chi-squared distribution.

PROOF. *Theorem 5.7, the lower bound of the density of the points denoting variance based flow features, when $L > 50$.*

Note that, when the degree of freedom is large, the chi-squared distribution can be approached by a Gaussian distribution. That is, when $L > 50$, the error of $\chi^2(L) \rightarrow \mathcal{N}(L, 2L)$ is small enough. Therefore, we can reformulate $E[R_{\text{Var}}]$ accordingly:

$$E[R_{\text{Var}}] = \frac{\sigma^2 N}{\sqrt{\pi}L} \int_{-\frac{\sqrt{L}}{2}}^{+\infty} (2\sqrt{L}t + L)e^{-t^2} \cdot \left[\frac{1+\text{erf}(t)}{2}\right]^{N-1} dt,$$

where $x = 2\sqrt{L}t + L$ and $dx = 2\sqrt{L}dt$. We observe that $\left[\frac{1+\text{erf}(t)}{2}\right]^{N-1}$ is monotonically increasing, which can be approached by a step function with a discontinuity located in $(1, +\infty]$, since N is large. Thus, we can obtain an upper bound for the range:

$$E[R_{\text{Var}}] \leq \frac{N\sigma^2}{\sqrt{\pi}L} \int_1^{+\infty} (2\sqrt{L}t + L)e^{-t^2}dt = \frac{N\sigma^2}{\sqrt{\pi Le}} + \frac{\sigma^2}{2}(1 - \text{erf}(1)).$$

Similarly, we can derive the lower bound of the density which is inversely proportional to σ^2. □

PROOF. *Theorem 5.8, the density of the points denoting variance based flow features, when $L = 2$.*

We obtain the closed-form expression for the density when $L = 2$, $f_{\chi^2(2)}(x) = \frac{1}{2}e^{-\frac{x}{2}}$, and $F_{\chi^2(2)}(x) = 1 - e^{-\frac{x}{2}}$. Specifically, we split the range into the left part $R^{(l)}$ and the right part $R^{(r)}$, which is the same as the proof for Theorem 5.5. According to the order statistic equation, we formulate the two parts of the range as:

$$E[R^{(r)}] = \frac{N\sigma^2}{4} \int_0^{+\infty} xe^{-\frac{x}{2}}\left[1 - e^{-\frac{x}{2}}\right]^{N-1} dx$$

$$= N\sigma^2 \int_0^1 -\ln u \cdot [1-u]^{N-1}du = \frac{N\sigma^2}{N+1} \cdot \sum_{i=1}^{N+1}\frac{1}{i},$$

where $x = -2\ln u, dx = -\frac{2}{u}du$, and $u \in [0, 1]$.

$$E[R^{(l)}] = \frac{\sigma^2}{4} \int_0^{+\infty} Nx \cdot \left[e^{-\frac{x}{2}}\right]^N dx$$

$$= \sigma^2 N \int_0^1 -\ln u \cdot u^{N-1}du = \frac{\sigma^2}{N}.$$

Finally, according to the equation $\mathrm{E}[R_{\mathrm{Var}}] = \mathrm{E}[R^{(r)}] - \mathrm{E}[R^{(l)}]$ and Definition 5.2, we derive the closed-form expression of the density in Theorem 5.8. □

B SUMMARY OF HYPER-PARAMETERS

We summarize the hyper-parameters of pVoxel in Table 7. Note that, we validate different settings by conducting cross validation to eliminate the impact of hyper-parameter bias [4].

Table 7: Hyper-parameter configuration of pVoxel.

Group	Hyper-Parameter	Description
Voxel Construction	ϵ	Edge length of the voxel.
	E	Minimum number of points.
Community	D	Definition of neighborhood.
Density Analysis	r	Radius of the balls.
	d_{Center}	Threshold for clustering.