RESEARCH-ARTICLE

Bringing IoT Intrusion Detection to the Edge

BARIKISU ASULBA, University of Porto, Porto, Porto, Portugal

PEDRO F SOUTO, University of Porto, Porto, Porto, Portugal

LUÍS TADEU ALMEIDA, University of Porto, Porto, Porto, Portugal

Open Access Support provided by:

University of Porto

Published: 11 December 2024

ICFNDS '24: The 8th International Conference on Future Networks & Distributed Systems
December 11 - 12, 2024
Marakech, Morocco

https://doi.org/10.1145/3726122.3726166
ISBN: 9798400711701

Bringing IoT Intrusion Detection to the Edge

Barikisu Asulba
CISTER, Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
up202103270@up.pt

Pedro F. Souto
CISTER, Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
pfs@fe.up.pt

Luis Almeida
CISTER, Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
lda@fe.up.pt

Abstract

Real-time network intrusion detection systems (NIDS) are essential for securing IoT networks. To enhance real-time detection, NIDS have to be deployed in the networks they protect, e.g., home networks or industrial networks, typically in edge equipment that has limited resources, e.g. the network router. Recent developments have shown the potential of using Machine Learning (ML), particularly Deep-Learning (DL) algorithms. However, DL is rather costly in computing resources. We present a novel approach based on classical ML methods suitable for anomaly detection (one-class methods), that shows a comparable detection capability with a fraction of the computing resource requirements of DL. We use the EDGE IoT/IIoT dataset to train five one-class ML algorithms. These include (SGD) One-Class Support Vector Machine, Elliptic Envelope, Isolation Forest, and Local Outlier Factor, each trained with different set sizes. We implement the models with Python for its wide dissemination and support, and we characterize the detection performance, the execution time and used memory in a setup based on a RaspberryPi that we consider representative of edge equipment. Our findings show that OCSVM generally outperforms the other approaches, including a DL model - MSM AE, in all metrics showing suitability for execution in the edge.

CCS Concepts

• **Real-time systems** → **real time detection systems**; • **Security and privacy** → *Intrusion detection systems*; • **Embedded systems** → Resource management; • **ML application systems** → ML IoT NIDS.

Keywords

Real-time systems, intrusion detection, security, embedded systems, resource management, machine learning, IoT, NIDS

ACM Reference Format:
Barikisu Asulba, Pedro F. Souto, and Luis Almeida. 2024. Bringing IoT Intrusion Detection to the Edge. In *The 8th International Conference on Future Networks & Distributed Systems (ICFNDS '24), December 11–12, 2024, Marakech, Morocco.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3726122.3726166

ICFNDS '24, Marakech, Morocco

ACM ISBN 979-8-4007-1170-1/24/12
https://doi.org/10.1145/3726122.3726166

1 Introduction

Internet of Things (IoT) and Industrial IoT (IIoT) bring significant benefits across industries, from industrial automation to smart homes. However, the network domain faces security challenges, as evidenced by a surge 107% in IoT-related attacks in early 2024 [6]. Routers, which account for 75% of IoT security breaches [11], highlight the need for robust security measures. Machine Learning-based Network Intrusion Detection Systems (ML-NIDS) have emerged as a promising solution, particularly when deployed at the edge for rapid local threat detection.

Although deep learning (DL) algorithms show potential in this domain [13], their high computational demands pose challenges for resource-constrained edge devices [25]. Making these models practical for edge deployment requires extensive compression and optimization - a process that is cost-intensive, demanding substantial resources and specialized expertise [29]. However, recent studies [10, 16] have shown that well-tuned classical machine learning (ML) models can achieve performance levels comparable to DL models in specific Intrusion Detection System (IDS) applications. Building on these findings, our work evaluates the performance of classical one-class machine learning (OC-ML) algorithms for intrusion detection in IoT and IIoT network.

OC-ML algorithms are particularly effective for detecting attacks using only normal traffic flows. This approach addresses critical challenges in network security, such as data scarcity and class imbalance. It is especially valuable in IoT and IIoT environments where: (1) collecting comprehensive datasets of malicious traffic is challenging due to the constantly evolving nature of cyber threats, and (2) detecting malicious traffic is inherently a generalization task, as the model is trained solely on normal traffic flows. These characteristics make OC-ML algorithms effective in identifying novel or unseen attacks, thereby overcoming a limitation of traditional supervised learning methods.

This study addresses two key questions:

(1) What is the performance of OC-ML for intrusion detection in edge environments and what are their resource consumption?
(2) How do OC-ML algorithms compare in performance to an OC- DL approach in resource-constrained edge environments?

We evaluated five OC-ML algorithms for IoT/IIoT network intrusion detection using a Raspberry Pi as an evaluation platform, comparable to routers deployed by Internet Service Providers [28]. With minimal fine-tuning, the One-Class Support Vector Machine (OCSVM) algorithm outperforms the OC-deep learning approach [3], achieving a higher detection rate (99.18% vs 93.16%) with a comparable false alarm rate (0.23% vs 0.18%). Moreover, it processes

2.41 times faster (27.81ms vs 67ms) on CPU compared to GPU, and handles over 3800 packets per second without drops. Key contributions:

(1) Demonstrating the efficiency of the OC-ML algorithms for edge-based IoT/IIoT intrusion detection.
(2) Analyze the impact of the size of the training data set on performance and resource use.
(3) Comparison of OC-ML with OC-DL models using consistent metrics.
(4) Providing insights into real-world constraints of edge-based ML-NIDS.

Exclusions: Development of novel ML algorithms, creation of new datasets, and testing on a physical IoT testbed setup.

2 Related Work

The increasing security and privacy concerns in IoT/IIoT environments have led to the development of various ML-NIDS. Several studies have proposed supervised learning techniques for the detection of intrusions in the IoT / IIoT network. [9, 12, 23, 26, 27]. These include self-knowledge distillation [27], feature grouping techniques [9], and hybrid CNN-LightGBM approaches [23]. However, these works often lack comprehensive resource usage metrics crucial for resource-constrained environments and are typically evaluated on high-end hardware, which is not representative of edge devices. For instance, Yang et al. [27] limit measurements to FLOPs, Khanday et al. [12] omit resource usage analysis, Zhao et al. [23] only report prediction time and He et al. [9] limited analysis of training time and memory usage.

In contrast, efforts to develop lightweight frameworks for edge-based intrusion detection, such as Wei et al. [26], have shown promise in testing on resource-constrained devices. Their two-stage deep learning approach achieved high accuracy on a Raspberry Pi 4B, demonstrating feasibility on actual edge hardware. However, it focused solely on packet-length features and reported only processing rate, still lacking a comprehensive resource analysis that includes memory usage. However, supervised methods face significant challenges, particularly in handling class imbalance in network datasets [5]. To address these challenges, including the detection of unknown attacks, researchers have explored one-class and unsupervised learning approaches. Borgioli et al. [3] proposed a one-class unsupervised deep auto-encoder, demonstrating potential in detecting novel attacks. However, their method faced real-time processing challenges on resource-constrained devices, with execution times limiting practical packet rates. Recent work by Safronov et al. [21] introduced a cloudless protection system combining rule-based filtering with AI-based threat detection. While they provided detailed resource usage analysis and combined rule-based and ML approaches, their work differs from ours in its focus. Our study specifically compares one-class ML models in terms of resources and performance for IoT/IIoT environments.

Traditional rule-based IDS implementations on edge devices [8] have revealed significant challenges, including high packet drop rates and variable CPU usage, highlighting the need for more efficient solutions. Despite advancements, critical gaps in lightweight ML for IoT/IIoT intrusion detection include:

(1) Limited testing on resource-constrained devices
(2) Insufficient resource usage analysis in "lightweight" implementations
(3) Few comparative studies of classical ML algorithms on edge devices

Our work addresses these gaps by evaluating one-class ML algorithms for IoT/IIoT edge environments. Using datasets from Borgioli et al. [3], we demonstrate robust detection with improved efficiency, providing comprehensive performance and resource usage measurements on a representative edge device.

3 Dataset

In the past decade, numerous publicly accessible datasets for ML-NIDs in IoT environments have been released, such as [2] and [15]. For this work, we chose the EDGE-IIoT dataset [7] due to its focus on Industrial IoT, various IoT devices, and a comprehensive set of recent network attacks. This dataset also allows for easy comparison with other ML approaches in related work. This source also provides a full description of the dataset and its generation process.

The dataset includes `pcap` files with full packet captures and CSV files with extracted features, covering various protocols in both the lower (TCP, UDP, ICMP) and upper layers (DNS, HTTP, MQTT). Our study focuses on TCP network traffic due to its prevalence in significant network attacks.

Table 1 summarizes the TCP flows used in this work, including normal traffic and various attack categories.

Table 1: Edge-IIoT Dataset TCP Flows

Category	Type of attack	TCP Flows
Normal	Various IoT Devices	666'862
DoS/DDoS	TCP SYN Flood	1'176'396
	HTTP Flood	23'627
Information Gathering	Port Scanning	9'987
	OS Fingerprinting	135
	Vulnerability Scanning	2'609
Injection	Cross-site Scripting (XSS)	1'149
	SQL Injection	4'382
	Uploading Attack	7'578
Malware	Backdoor Attack	827
	Password Cracking	94'498
	Ransomware	570

4 Methodology

Our approach integrates several stages—feature extraction, data preprocessing, model training, and evaluation—to develop a robust intrusion detection system for IoT networks. We outline our methodology below, emphasizing the rationale behind each stage.

4.1 Feature Extraction and Data Preprocessing

We adopt a packet flow-based approach, similar to [7], as it allows for a more comprehensive analysis of network behavior compared to packet-based methods. This approach differs from [3], where packets are directly submitted to a DL autoencoder. Our method enables the capture of temporal and behavioral patterns within network flows, which is crucial for detecting complex attack patterns.

4.1.1 Feature Extraction. We employ `tstat` [1], a robust network traffic analysis tool, to process `pcap` files and generate detailed TCP connection features. Configured to produce the Core TCP Set, `tstat` extracts 44 basic features critical for analyzing both complete and incomplete TCP connections. This comprehensive feature set captures a wide range of network behaviors, including packet counts, bytes transmitted, retransmission patterns, and time-related metrics (e.g., first and last packet times, connection duration). These features are particularly effective in highlighting deviations from normal behavior, such as high counts of SYN segments or unusual retransmission patterns, which may indicate ongoing attacks. The chosen features enable the detection of various attack types, including those involving incomplete connections like TCP SYN floods.

4.1.2 Data Preprocessing. Our preprocessing strategy aims to refine the set of features while preserving the most relevant information for attack detection. The process involves:

(1) Concatenating `log_tcp_complete` and `log_tcp_nocomplete` logs from `tstat` .
(2) Removing potentially identifying information (IP addresses, ports) to ensure privacy.
(3) Eliminating features with zero mean or standard deviation to focus on informative attributes.
(4) Applying Min-Max Normalization to Standardize Feature Scales.

Table 2 details the TCP flow features used in our models."C" and "S" denote the client (initiator of the handshake) and the server (recipient of the handshake), respectively.

4.2 Model Selection and Configuration

We selected five algorithms representing diverse approaches to one-class classification, covering density-based, boundary-based, and tree-based methods. These algorithms were chosen based on their effectiveness in anomaly detection.

- Local Outlier Factor (LOF) [4] detects anomalies based on local density differences.
- Elliptic Envelope (Elliptic Env.) [20] identifies outliers as data points outside a fitted "ellipse."
- One-Class Support Vector Machine (OCSVM) [22] separates normal data from outliers using a hyperplane.
- Stochastic Gradient Descent (SGD) OCSVM[24] approximates the OCSVM solution using stochastic gradient descent.
- Isolation Forest (iForest) [14] isolates data points using binary trees.

To demonstrate the effectiveness of minimally tuned classical ML models and focus on real-world applicability in resource-constrained IoT/IIoT environments, we used default parameters for most settings. We empirically determined the optimal decision threshold for each algorithm through a controlled comparison, keeping all other parameters at their default values. This approach allowed us to fine-tune the model sensitivity to anomalies without extensive parameter tuning. Specifically, contamination was set to **0.05** for LOF, Elliptic Env., and iForest; nu was set to **0.001** for OCSVM and **0.01** for SGD-OCSVM.

4.3 Experimental Setup

Dataset Splitting: The Edge-IIoT dataset (666'862 normal TCP flows) was split into 600'000 training and 66'862 testing flows, maintaining device proportions. The training set was divided into 12 chunks of 50'000 flows each, preserving the device proportions.

Training Subsets: To evaluate dataset size effects, we created subsets of 1'000, 5'000, 10'000, and 50'000 flows from each chunk. This allows us to assess the impact of the data volume and determine the minimal data needed for effective performance. Each algorithm was trained on these subsets (240 models total).

Testing Set: Comprised of 66'862 normal flows and all malign TCP flows.

5 Evaluation

Models were trained on a desktop computer with `Scikit-learn` [18], and their resource consumption was evaluated on a Raspberry Pi 3 Model B+ Rev 1.3.

5.1 Comparison of Evaluation Platform

This study uses the Raspberry Pi 3 Model B+ as a representative network edge device. To contextualize its capabilities, we compare it with a typical home WRT router, as shown in Table 3. We focus on WRT routers because manufacturers often do not disclose detailed hardware specifications for proprietary devices, and platforms like OpenWRT provide reliable information on the hardware used in these routers, offering a more transparent comparison. The Raspberry Pi 3 Model B+ utilizes a 16 GB SD card for storage, significantly exceeding the flash memory capacity (typically 128 to 512 MB). However, many routers include SD card slots, which facilitate storage expansion, making deployment on these devices possible when additional storage is necessary. Also, compared to SoC routers like the Qualcomm IPQ8071A, both the Raspberry Pi 3B+ and the IPQ8071A use quad-core ARM Cortex-A53 processors, based on the ARMv8 architecture. However, the Raspberry Pi 3B+ runs at 1.4 GHz, while the IPQ8071A operates at up to 1.0 GHz. This shows that the Raspberry Pi offers computing power similar to that of typical routers. Thus, it serves as a suitable network edge device for testing OC-ML algorithms, offering valuable insights for deployment on routers. Table 3 presents a comparison between the Raspberry Pi 3B+ and typical home router.

5.2 Metrics

Our evaluation focused on both detection accuracy and resource consumption during inference:

- **Detection Accuracy:**
  - **True Positive Ratio (TPR) (Recall):** Proportion of attack flows correctly detected, ideally 1.
  - **False Positive Ratio (FPR):** Rate of benign flows misidentified as attacks, ideally 0.
- **Resource Consumption:**
  - **Inference Time per Flow:** Time taken to classify one flow.
  - **Memory Footprint:** Memory used during execution, excluding shared memory.

Table 2: Features Used in Intrusion Detection

C2S/S2C	feature	Unit	Description
3/17	packets	-	Total number of packets observed from the C2S/S2C
5/19	ACK sent	-	Number of segments with the ACK field set to 1
6/20	PURE ACK sent	-	Number of segments with ACK field set to 1 and no data
7/21	unique bytes	bytes	Number of bytes sent in the payload
8/22	data pkts	-	Number of segments with payload
9/23	data bytes	bytes	Number of bytes transmitted in the payload, retransmissions
10/24	rexmit pkts	-	Number of retransmitted segments
11/25	rexmit bytes	bytes	Number of retransmitted bytes
13/27	SYN count	-	Number of SYN segments observed (including retransmissions)
14/28	FIN count	-	Number of FIN segments observed (including retransmissions)
18	RST sent	0/1	RST segment has been sent by the server
29	First time abs	ms	Flow first packet absolute time (epoch)
30	Last time abs	ms	Flow last segment absolute time (epoch)
31	Completion time	ms	Flow duration from first packet to last packet
32	C first payload	ms	Client first segment with payload since the first flow segment
33	S first payload	ms	Server first segment with payload since the first flow segment
34	C last payload	ms	Client last segment with payload since the first flow segment
35	S last payload	ms	Server last segment with payload since the first flow segment
36	C first ack	ms	Client first ACK segment (without SYN) since the first flow
37	S first ack	ms	Server first ACK segment (without SYN) since the first flow
42	Connection type	-	Bitmap stating the connection type as identified by TCPL7

Table 3: Specification Comparison: Raspberry Pi 3B+ vs. Typical Home Routers

Feature	Raspberry Pi 3B+	Typical Home Router[17]	Qualcomm IPQ8071A[19]
Processor	Quad-core ARM Cortex-A53	Varies (e.g., Qualcomm,)	Quad-core ARM Cortex-A53
Clock Speed	600 MHz - 1.4 GHz (variable)	1.2 GHz - 2.5 GHz (model-dependent)	Up to 1 GHz
CPU Cores	4	2 - 4 (typical range)	4
RAM	920 MB	128 MB - 1 GB	DDR3L/DDR4
Storage	16 GB SD card	128 MB - 512 MB Flash	NOR/NAND Flash
Wi-Fi	2.4GHz and 5GHz 802.11.b	Up to 802.11ax (Wi-Fi 6)	Wi-Fi 6 (up to 4.8 Gbps)
Ethernet	1x Gigabit port	2 - 8 Gigabit ports	2 - 8 Gigabit ports
USB Ports	4x USB 2.0	0 - 4 (USB 2.0 or 3.0)	USB 3.0
OS	Raspbian GNU/Linux 12	Proprietary firmware or OpenWrt	Proprietary firmware
Primary Use	General computing, IoT projects	Networking, device management, ISP connectivity	Networking, gateways, access points

– **Memory Size:** Size of the serialized trained model.

5.3 Detection Accuracy

Table 4 shows the TPR of the different models, for the four classes of network attacks with TCP flows, namely Distributed Denial of Service, Information Gathering, Malware and Injection Attacks, respectively (Table 1). For each algorithm, we used four different training set sizes. For each combination of algorithm and training set size, we generated 12 models using disjoint training sets. Therefore, the values presented in these tables are the averages of the TPR of each of these 12 models. All algorithms, with the exception of SGD-OCSVM, have very high TPRs regardless of the attack class and the specific attack in a class.

The results for SGD-OCSVM are very uneven. For example, whereas it has a TPR of 100% for port scanning flows, its TPR is very close to 0% for vulnerability scanning flows. Also, it appears to be very sensitive to the size of the training set, but not always with a defined trend. For example, for TCP SYN Flood attacks, the maximum average TPR is 83.33% for the models trained with 1'000 flows, whereas the minimum average TPR is 16.67% for the models trained with 10'000 flows. We suspect that this is because of limited hyper-parameter tuning; as we have mentioned above, the only hyper-parameter we tuned was nu.

For the other models, we also observe some fluctuations on the average TPR with the size of the training set sizes, depending on the attacks. In general, even models trained with datasets with 1'000

flows provide a high average TPR, but training datasets with 5'000 flows have the highest average TPR with just a few exceptions.

Based only on the average TPR, there is no clear winner. However, for NIDS the false alarm ratio, or FPR, is also a very important metric. Since we can expect that most of the time the network is not under attack, if the FPR is different from 0, the NIDS will report frequent false alarms. This can lead operators to neglect an alarm for a real attack. Table 5 shows the average False Positive Ratio for the different algorithms and training dataset sizes.

Whereas for Isolation Forest and OCSVM, and to a less extent Elliptic Envelope, the FPR tends to decrease when increasing the training dataset size, for Local Outlier Factor the FPR increases with the size of the training dataset. On the other hand, for SGD-OCSVM there is no clear trend.

In any case, except for LOF with training datasets of 1'000 flows, Local Outlier Factor, Elliptic Envelope and Isolation Forest all have FPR above 2%, which is too high for the reasons already mentioned. SGD-OCSVM and OCSVM models, except OCSVM models trained with datasets of only 1'000 flows, exhibit FPR below 1%.

Taking into account both metrics, OCSVM is the clear winner among these 5 algorithms. It combines a very high TPR with a very low FPR. According to our results, a training dataset of 5'000 flows is the one that shows a better balance between these two metrics. Increasing the training dataset size reduces marginally the FPR, by less than 0.2%, but it also reduces the TPR by a few percent for some attacks. Decreasing the training dataset size maintains a high TPR, but increases the FPR to over 2%. In general, whether this is acceptable or not, may depend on the application scenario.

5.4 Resource Consumption

5.4.1 Inference times. Inference times on the Raspberry Pi were recorded using Python's `time` module for each of the 12 models across different training dataset sizes. Each model was tested with 20,000 flows (both attack and normal samples), resulting in 240,000 data points in total. Figure 1 displays two graphs for two different algorithms, namely OCSVM on the left and Elliptic Env. on the right. While OCSVM is our best model, the Elliptic Env. is our second-best, shown here for comparison, with a perfect TPR and significantly lower resource requirements, but a higher FPR. The box plots illustrate the inference time as a function of the training dataset size, using a linear scale on both axes, and each box plot encompasses all values for the 12 models. These plots show that the inference time distribution of Elliptic Env. is essentially independent of the training dataset size while OCSVM shows a slightly increasing inference time as the training dataset size grows. Except for OCSVM trained with 50'000 flows, both algorithms have an average inference time per flow below 10 ms.

5.4.2 Memory Usage- Unique Set Size (USS). The memory footprint at runtime was monitored using the `smem` tool, which recorded process memory usage on the Raspberry Pi every 5 seconds during model prediction. Figure 2 shows two graphs displaying the models memory footprint in USS against the training dataset size for our two best approaches, namely OCSVM on the left and Elliptic Envelope on the right. Similarly to the inference time, OCSVM exhibits a slight increase in memory usage with growing dataset size, overcoming 100 MB when training with 50,000 samples. Conversely, Elliptic Envelope maintains a consistent average close to 94 MB.

5.4.3 Model Size. Table 6 illustrates the memory footprint of OCSVM and Elliptic Envelope models. OCSVM's memory usage scales linearly with dataset size, ranging from 136.9 KB to 6.86 MB, reflecting its storage of support vectors. In contrast, Elliptic Envelope maintains a constant 16.2 KB footprint across all dataset sizes, as it only stores parameters of the elliptical decision boundary. This comparison highlights the trade-off between model complexity and memory efficiency, a crucial consideration for resource-constrained IoT environments.

5.4.4 Feature Extraction Resource Usage. Since our models are flow-based and use the core TCP features of `tstat` , their deployment will also require the deployment of `tstat` on the same platform for capturing packets and extracting the core TCP-flow features. Therefore, we have also run some experiments to evaluate both the runtime per flow processing time of `tstat` as well as its memory usage on the Raspberry Pi.

Table 7 shows some features of the `pcap` files and the corresponding resource consumption by `tstat` upon processing these files. The `pcap` files shown include both packet traces for some of the attacks listed in Table 1 and packet traces for normal traffic generated by some sensors. The Elapsed Time (s) is the difference in seconds between the timestamps of the last and of the first packet of the respective file. The runtime was measured using the Unix `time` command recording, which retrieves the actual time taken from the start to the end of the `tstat` execution. This includes reading the `pcap` file from disk. The time required to process a packet or a flow is actually an average time and was derived by dividing the measured run-time by the total number of packets and by the total number of flows. As shown, the average runtime per flow varies significantly and so does the average runtime per packet. We can also observe that there is no correlation between the runtime per flow and the type of trace.

Table 7 also shows `tstat` memory footprint in USS, which was also measured with `smem`. This was done by recording the statistics every second while running `tstat` for each `pcap` file. Unlike the run-time for processing each flow, the USS was the same for all executions of `tstat` , and it is independent of the size of the `pcap` file. However, the memory footprint of the models exceeds that of `tstat` by approximately 60%.

5.5 Comparison with Other Works

The Edge-IIOT dataset was used by at least two other works to evaluate NIDS based on different ML-Algorithms.

The first one [7], by the dataset creators, uses four conventional classification ML algorithms, namely Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM) and K-Nearest Neighbor (KNN). In addition, they used a Deep Neural Network (DNN)-based classifier, using both a centralized and federated learning. We will not discuss the latter. Like our models, their models are flow-based, although they used tools different from `tstat` for extraction of the flows features from the `pcap` files. For training these models, the authors used a supervised learning approach. Furthermore, they

Table 4: Average True Positive Ratio for Various Attack Types

Model	Size (k)	DDoS		Information Gathering			Malware			Injection		
		SYN	HTTP	Port	OS	Vuln.	Back.	Pass.	Rans.	XSS	SQL	Upload
Elliptic Env.	1	0.9167	0.959	0.9167	0.9167	1.0000	0.9229	0.9976	0.9348	0.9473	1.0000	0.9379
	5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	10	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	50	1.0000	0.9738	1.0000	1.0000	0.9978	1.0000	0.9750	1.0000	0.9306	0.9627	0.9793
iForest	1	0.9167	1.0000	1.0000	0.9167	0.9222	1.0000	0.9863	1.0000	0.9927	0.9194	0.9971
	5	1.0000	1.0000	1.0000	1.0000	0.9469	1.0000	0.9955	1.0000	0.9961	0.9509	1.0000
	10	1.0000	1.0000	1.0000	1.0000	0.9224	1.0000	0.9924	1.0000	0.9993	0.8934	0.9994
	50	1.0000	1.0000	1.0000	1.0000	0.9410	1.0000	0.9965	1.0000	0.9994	0.9603	1.0000
LOF	1	0.8334	0.8773	0.9167	0.9167	1.0000	0.9167	0.9792	0.9164	1.0000	0.9994	0.9375
	5	1.0000	0.9862	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	10	1.0000	0.9336	1.0000	1.0000	1.0000	1.0000	1.0000	0.9996	1.0000	1.0000	1.0000
	50	0.8334	0.9564	0.9167	1.0000	1.0000	1.0000	0.9999	1.0000	0.9973	0.9994	1.0000
OCSVM	1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	10	1.0000	1.0000	1.0000	1.0000	0.9991	1.0000	0.9825	1.0000	0.9731	0.9994	0.9998
	50	1.0000	1.0000	1.0000	1.0000	0.9982	1.0000	0.9650	1.0000	0.9462	0.9989	0.9997
SGD_OCSVM	1	0.8333	0.7801	1.0000	1.0000	0.0000	1.0000	0.2492	0.9993	0.0000	0.0068	0.7461
	5	0.2735	0.8352	1.0000	1.0000	0.0000	1.0000	0.2492	1.0000	0.0000	0.0068	0.7461
	10	0.1667	0.8400	1.0000	1.0000	0.0001	1.0000	0.2317	1.0000	0.0001	0.0062	0.7463
	50	0.4167	0.8505	1.0000	1.0000	0.0428	1.0000	0.2220	1.0000	0.1097	0.0978	0.7879

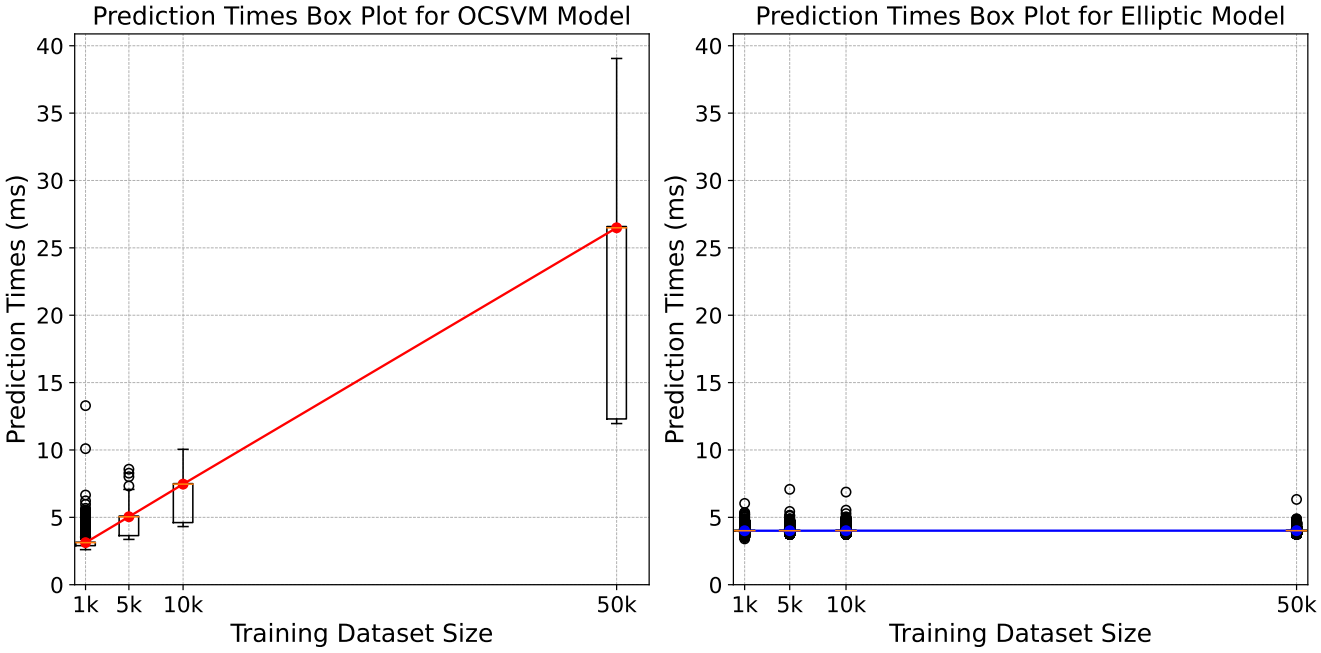

Figure 1: Boxplot of inference times per flow.

varied the number of classes: 2, for normal vs. attack, 6, normal vs. each of the 5 attack classes, and 15, normal vs each of the 14 specific attacks. Their work does not consider computational resource consumption at inference time.

The second work [3] uses 3 different neural network autoencoder architectures namely, 1D Convolutional NN AE, Long-Short Term Memory (LSTM) AE and a novel AE architecture they named Multi-State Memory (MSM), to overcome some limitations they observed in the LSTM AE model. Here, we will focus on the MSM AE because the others have a very high FPR (18.49% and 17.67%, respectively). Unlike the models in our work and in [7], these models are packet-based, i.e. they take as input packets and classify each

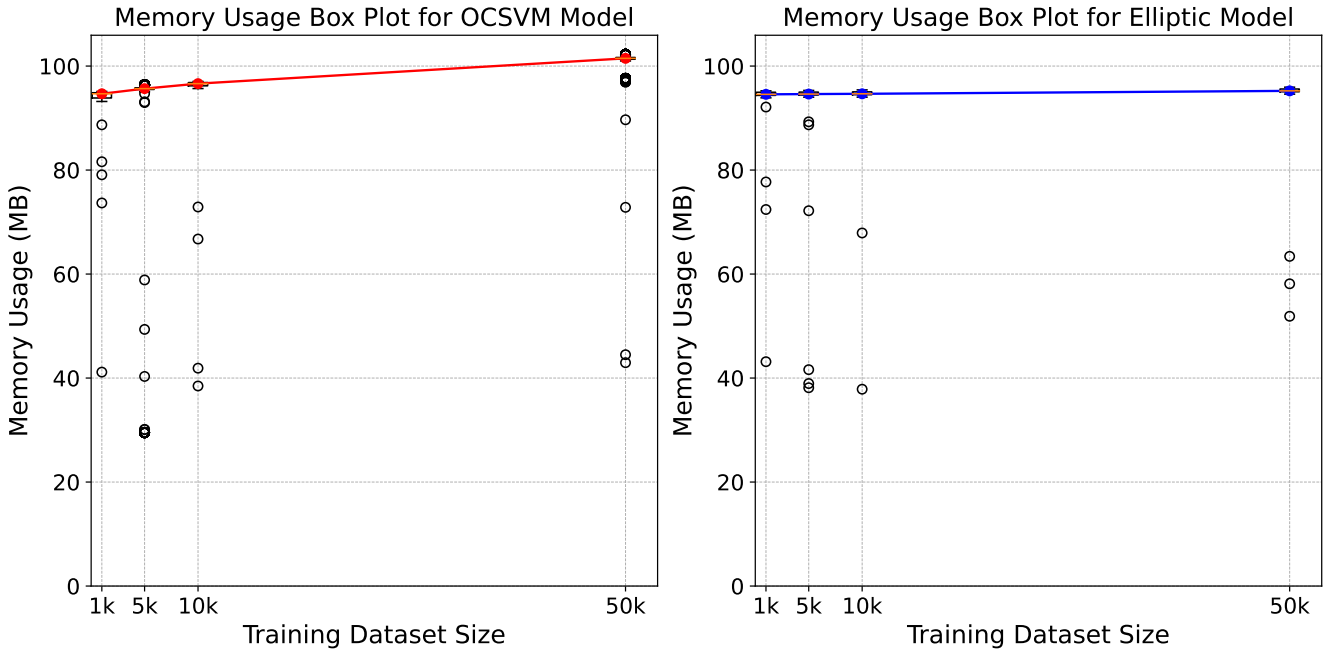

Figure 2: Boxplot of models memory footprint per training dataset size.

Table 5: Average False Positive Ratio on Normal Flows

Model	1k	5k	10k	50k
Elliptic Env.	0.0505	0.0473	0.0411	0.0413
iForest	0.0562	0.0576	0.0527	0.0507
LOF	0.0068	0.2474	0.3325	0.465
OCSVM	0.0256	0.0038	0.0027	0.0023
SGD_OCSVM	0.006	0.0033	0.0006	0.0059

Table 6: Model Memory Size (KB) vs. Training Dataset Size

Model	1k	5k	10k	50k
OCSVM	136.9	672.2	1331.2	6860.8
Elliptic	16.2	16.2	16.2	16.2

packet as either normal or attack. The model was trained using only normal traffic, but attack traffic was also used for hyperparameter tuning. This work did not consider the memory footprint of the models, but it measured the inference time on an NVIDIA Jetson AGX Orin Developer Kit, an embedded platform, both using and not using its GPU.

We will focus on a quantitative comparison with the results presented in [3], not only because that work also targets embedded platforms, but also because the work in [7] does not provide detailed results for one-class models, and using the results for the models with more classes would not be fair, as it is harder to classify a flow as belonging to a class of attacks or to a specific attack than whether it is normal or not. Furthermore, we compare only with the OCSVM models created with training dataset sizes of 5K, 10K and 50K flows, which, as discussed earlier, are the models that provide the best prediction performance.

Table 9 shows the FPR, i.e. the fraction of normal flows misclassified as attack flows, obtained with the models under consideration. These results show that using supervised learning reflects on an improvement of the FPR (the closer to 0 the best). For the models that were not trained with attack traffic, we can see that OCSVM is competitive with MSM AE, especially for models trained with larger number of flows. Actually, the performance of the OCSVM is better than it appears from these numbers, because the MSM AE classifies packets, and each TCP flow has several packets. Therefore, we can expect MSM AE to generate more "false alarms" than OCSVM, even for the model trained with 5K flows. On the other hand, the OCSVM models handle only TCP-flows, whereas the MSM AE is transport protocol agnostic.

Table 10 shows the TPR, i.e. the detection-ratio of attack flows, for the different models under consideration, for each of the attacks using TCP. This table shows that, regardless of the training dataset size, the TPR of the OCSVM models are very close to that of the MSM-AE model, if not higher.

Finally, Table 11 shows the average inference times of the algorithms under consideration. Note that the times reported for the OCSVM algorithms, already include the time taken by `tstat` to process the flows. For the MSM AE model, we include the values for both with and without GPU, at the maximum power level, reported in [3], which were measured using an NVIDIA Jetson AGX Orin Developer Kit.

As shown in Table 7, the time it takes `tstat` to process a flow is highly variable, therefore in this comparison we are using a value

Table 7: Runtime resource consumption of `tstat` on Raspberry Pi

Type	Pcap File				Resource Consumption			
	Size (MB)	Packets	Flows	Elapsed (s)	Runtime (s)	Time/ Packet (ms)	Time/ Flow (ms)	USS (MB)
Uploading	5.49	37'636	7'586	505.32	3.50	0.093	0.46	22.32
Backdoor	6.69	24'590	865	328'537.97	54.66	2.23	63.22	22.32
DDoS HTTP	28.31	229'122	23'634	271.11	5.32	0.023	0.23	22.32
Password	194.6	1'053'485	94'780	68'505.64	358.21	0.340	3.78	22.32
Port Scanning	1.8	21'583	10'082	9'999.72	48.77	2.26	4.84	22.32
Ransomware	2.8	10'525	630	116'550.44	557.12	52.93	884.32	22.32
XSS	2.1	16'156	1'235	8'007.29	37.42	2.32	30.34	22.32
Vulnerability	140.4	265'706	2'697	2'394.66	15.17	0.057	5.63	22.32
SQL Injection	8.17	51'213	4'391	505.12	3.85	0.075	0.88	22.32
DDoS TCP	282.61	2'020'047	1'176'416	588.01	98.26	0.049	0.080	22.32
OS Fingerprint	0.11	1'094	140	821.59	4.12	3.77	29.43	22.32
Heart Rate	12.84	170'263	10'426	697.39	11.56	0.068	1.11	22.32
Distance	85.42	1'143'891	71'896	1'612.04	26.36	0.023	0.374	22.32
Flame Sensor	79.69	1'070'510	67'257	1'437.10	22.80	0.021	0.337	22.32
IR Receiver	96.91	1'308'204	82'434	1'826.33	30.03	0.023	0.361	22.32
pH Value	56.55	759'319	45'189	4'467.28	33.75	0.044	0.832	22.32
Soil Moisture	85.65	1'200'376	70'622	8'182.12	58.72	0.049	0.829	22.32
Sound Sensor	112.12	1'513'312	94'869	1'936.78	35.14	0.023	0.369	22.32
Water Level	172.48	2'302'250	144'889	6'703.43	3'239.43	1.41	22.35	22.32

Table 8: Aggregated Runtime Resource Consumption of `tstat` on Raspberry Pi

Category	Total Size (MB)	Total Packets	Total Flows	Total Runtime (s)	Avg Time/ Packet (ms)	Avg Time/ Flow (ms)	Avg USS (MB)
Attack	673.08	3'731'157	1'322'456	1'186.40	0.3183	0.8978	22.32
Normal	701.66	9'468'125	587'582	3'457.79	0.3653	5.8848	22.32
All	1'374.74	13'199'282	1'910'038	4'644.19	0.3521	2.4305	22.32

Table 9: False Positive Ratio

Model	OCSVM(5K)	OCSVM(10K)	OCSVM(50K)	MSM AE [3]	DNN [7]
FPR	0.0038	0.0027	0.0023	0.0018	0

Table 10: True Positive Ratio

Model	OCSVM(5K)	OCSVM(10K)	OCSVM(50K)	MSM AE [3]
TCP SYN Flood	1.0000	1.0000	1.0000	0.9999
HTTP DDoS	1.0000	1.0000	1.0000	0.9967
Port Scanning	1.0000	1.0000	1.0000	0.5565
OS Fingerprint.	1.0000	1.0000	1.0000	0.8624
Vulnerab. Scan.	1.0000	0.9991	0.99982	0.9986
Backdoor	1.0000	1.0000	1.0000	0.9837
Password Crack.	1.0000	0.9825	0.9650	0.9982
Ransomware	1.0000	1.0000	1.0000	0.9386
XSS	1.0000	0.9731	0.9462	0.9332
SQL Injection	1.0000	0.9994	0.9989	0.9993
Uploading	1.0000	0.9998	0.9997	0.9806

Table 11: Inference Time

Model	OCSVM(5K)	OCSVM(10K)	OCSVM(50K)	MSM-CPU	MSM-GPU
Time	6.59 ms	8.52 ms	27.81 ms	256 ms	67 ms

of 2 ms, which is the global average of the times per flow while processing all pcap files shown in Table 8.

Although the values in Table 7 were not measured on the same platform, both use an ARM processor, and the NVIDIA JETSON uses a more modern one, an A-78 vs an A-53. The authors of [3] do not provide the details about the specific Developer Kit. But even assuming the least powerful kit, only its CPU is more capable than that of the A-53 and uses a higher clock rate. Thus, clearly this table shows that the inference time of the OCSVM algorithms, even implemented in Python, are much shorter than that of MSM AE, even using a GPU.

Because the OCSVM algorithms are flow-based whereas the MSM AE are packet-based, these numbers need to be carefully analyzed. It is possible that the MSM AE, in spite of taking more time to classify a packet than the OCSVM algorithms to classify a flow, will have a lower response time in reporting an attack. The flow detection is highly dependent on flow features such as the respective packet rate and number of packets in the flow. On the other hand, the throughput of the MSM AE is very low, namely less than 20 packets per second, using the GPU with full power, whereas the OCSVM algorithms, even implemented in Python, are able to process about 100 flows per second, in a less powerful and less energy-consuming platform. It is unclear whether the

implementation of MSM AE is able to process several packets in parallel, e.g. by pipelining the different processing steps. Even if that is possible, it is still unclear whether it will be enough to close this wide gap in the near future, particularly if optimized implementations of the conventional ML models are used, e.g., written on a compiled language such as C.

6 Conclusions

Network Intrusion Detection Systems in IoT/IIoT ecosystems is becoming crucial for assuring secure operation. Machine Learning models are revealing to be an adequate option given their robustness and capacity to generalize, thus being capable of detecting new attacks. Previous works explored the use of supervised learning methods and Deep Learning models. Conversely, we use traditional one-class ML algorithms used in anomaly detection and we show that we can achieve a similar or higher detection performance while using a fraction of the computing resources. We developed 240 one-class ML models based on five different algorithms to detect network attacks, by training them with datasets of different sizes with only normal traffic flows of the Edge-IIoTset dataset. The evaluation of these models using not only normal traffic flows, but also network attack flows, both from Edge-IIoTset. showed that the OCSVM models, even when trained with only 5'000 flows, had a very high true positive ratio, i.e. are able to detect all new attack flows of most of the attacks and above 90% for all the remaining attacks, with a low false positive ratio, i.e. these models misclassified as attacks less than 1% of the normal flows. This prediction performance is on par with that of a novel autoencoder architecture presented in [3]. Furthermore, the time these models take to classify a flow, even when implemented in Python, including the time needed to extract the flow features from the captured packets, is less than 20% of the time the auto-encoder takes to process a packet in its fastest implementation.

While our findings are promising, we acknowledge certain limitations. Using a Raspberry Pi, while representative of edge devices, may not fully reflect the performance characteristics of home routers in a real scenario. Furthermore, while the Edge-IIoTset dataset provided a solid foundation for our study, it may not encompass the full diversity of real-world IoT environments and emerging threats. Additionally, it is important to note that the performance of any AI-based solution, including ours, may vary when applied to datasets that differ significantly from the training data, which is a common challenge in machine learning. Despite these limitations, our findings show that with current technology, conventional anomaly detection OC-ML algorithms outperform OC-DL models running on network edge devices. Our methodology can be applied to various IoT/IIoT network environments, allowing for broader applicability.

Acknowledgments

This work was financially supported by: Base Funding (UIDB/04234/2020) and Programatic Funding (UIDB/04234/2020) of the Research Centre in Real-Time and Embedded Computing Systems (CISTER) funded by national funds through the FCT/MCTES (PIDDAC).

References

[1] [n. d.]. Tstat. http://tstat.polito.it/. Accessed on Feb 1, 2023.
[2] Josue Genaro Almaraz-Rivera, Jesus Arturo Perez-Diaz, Jose Antonio Cantoral-Ceballos, Juan Felipe Botero, and Luis A. Trejo. 2022. LATAM-DDoS-IoT dataset. (2022). https://doi.org/10.21227/rwtj-dd43
[3] N. Borgioli, L. T. X. Phan, F. Aromolo, A. Biondi, and G. Buttazzo. 2023. Real-time packet-based intrusion detection on edge devices. *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023* (2023). https://doi.org/10.1145/3576914.3587551
[4] M. Breunig, H. Kriegel, R. Ng, and J. Sander. 1999. OPTICS-OF: identifying local outliers. In *Principles of Data Mining and Knowledge Discovery*. 262–270. https://doi.org/10.1007/978-3-540-48247-5_28
[5] Nathan Dai and Suleyman Uludag. 2024. Performance Tradeoff in ML-Based Intrusion Detection Systems: Efficacy vs. Resource Usage. In *2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*. IEEE, 1030–1031.
[6] Digit.fyi. 2024. *Malware and IoT Attacks See Huge Rise in H1 2024.* Digit.fyi. https://www.digit.fyi/malware-and-iot-attacks-see-huge-rise-in-h1-2024/
[7] Mohamed Amine Ferrag, Othmane Friha, Djallel Hamouda, Leandros Maglaras, and Helge Janicke. 2022. Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning. *IEEE Access* 10 (2022), 40281–40306. https://doi.org/10.1109/ACCESS.2022.3165809
[8] Toghrul Garalov and Mohammed Elhajj. 2023. Enhancing IoT Security: Design and Evaluation of a Raspberry Pi-Based Intrusion Detection System. In *2023 International Symposium on Networks, Computers and Communications (ISNCC)*. 1–7. https://doi.org/10.1109/ISNCC58260.2023.10323656
[9] Mingshu He, Yuanming Huang, Xinlei Wang, Peng Wei, and Xiaojuan Wang. 2024. A Lightweight and Efficient IoT Intrusion Detection Method Based on Feature Grouping. *IEEE Internet of Things Journal* 11, 2 (2024), 2935–2949. https://doi.org/10.1109/JIOT.2023.3294259
[10] Haochen Hua, Yutong Li, Tonghe Wang, Nanqing Dong, Wei Li, and Junwei Cao. 2023. Edge Computing with Artificial Intelligence: A Machine Learning Perspective. *ACM Comput. Surv.* 55, 9, Article 184 (jan 2023), 35 pages. https://doi.org/10.1145/3555802
[11] IoT Business News. 2024. *New Report on IoT Security Underscores the Current Risk of Unsecured Devices and Equipment.* IoT Business News. https://iotbusinessnews.com
[12] Shahbaz Ahmad Khanday, Hoor Fatima, and Nitin Rakesh. 2023. Implementation of intrusion detection model for DDoS attacks in Lightweight IoT Networks. *Expert Systems with Applications* 215 (2023), 119330. https://doi.org/10.1016/j.eswa.2022.119330
[13] Jan Lánský, Saqib Ali, Mokhtar Mohammadi, Mohammed Kamal Majeed, Sarkhel H.Taher Karim, Shima Rashidi, Mehdi Hosseinzadeh, and Amir masoud Rahmani. 2021. Deep Learning-Based Intrusion Detection Systems: A Systematic Review. *IEEE Access* 9 (2021), 101574–101599. https://api.semanticscholar.org/CorpusID:236479767
[14] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *IEEE International Conference on Data Mining 2008 (ICDM 08)*. https://doi.org/0.1109/ICDM.2008.17
[15] Nour Moustafa. 2019. ToN_IoT datasets. IEEE Dataport. https://doi.org/10.21227/fesz-dm97 Accessed on: Feb 1, 2023.
[16] M. G. Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. 2021. Machine Learning at the Network Edge: A Survey. *ACM Comput. Surv.* 54, 8, Article 170 (oct 2021), 37 pages. https://doi.org/10.1145/3469029
[17] OpenWrt Wiki. 2024. Table of Hardware (Extended). https://openwrt.org/toh/views/toh_extended_all. Accessed: 15-Oct-2024.
[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[19] Qualcomm. n.d.. Qualcomm Networking Pro 600 Platform. https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/qualcomm_networking_pro_600_platform_final.pdf Accessed: 2024-10-18.
[20] P. J. Rousseeuw and K. V. Driessen. 1999. A fast algorithm for the minimum covariance determinant estimator. *Technometrics* 41 (1999), 212. Issue 3. https://doi.org/10.2307/1270566
[21] Vadim Safronov, Anna Maria Mandalari, Daniel J. Dubois, David Choffnes, and Hamed Haddadi. 2024. SunBlock: Cloudless Protection for IoT Systems. In *Passive and Active Measurement*, Philipp Richter, Vaibhav Bajpai, and Esteban Carisimo (Eds.). Springer Nature Switzerland, Cham, 322–338.
[22] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. 2001. Estimating the Support of a High-Dimensional Distribution. *Neural Computation* 13, 7 (2001), 1443–1471. https://doi.org/10.1162/089976601750264965
[23] Guo sheng Zhao, Yang Wang, and Jian Wang. 2023. Intrusion Detection Model of Internet of Things Based on LightGBM. *IEICE Trans. Commun.* 106 (2023),

622–634. https://api.semanticscholar.org/CorpusID:245499868

[24] C. Shieh, T. Nguyen, C. Chen, and M. Horng. 2022. Detection of unknown ddos attack using reconstruct error and one-class svm featuring stochastic gradient descent. *Mathematics* 11 (2022), 108. Issue 1. https://doi.org/10.3390/math11010108

[25] Md. Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I. Morshed. 2023. Efficient Acceleration of Deep Learning Inference on Resource-Constrained Edge Devices: A Review. *Proc. IEEE* 111 (2023), 42–91. https://api.semanticscholar.org/CorpusID:254706813

[26] Chongbo Wei, Gaogang Xie, and Zulong Diao. 2023. A lightweight deep learning framework for botnet detecting at the IoT edge. *Computers & Security* 129 (2023), 103195. https://doi.org/10.1016/j.cose.2023.103195

[27] Shuo Yang, Xinran Zheng, Zhengzhuo Xu, and Xingjun Wang. 2023. A Lightweight Approach for Network Intrusion Detection Based on Self-Knowledge Distillation. In *ICC 2023 - IEEE International Conference on Communications*. 3000–3005. https://doi.org/10.1109/ICC45041.2023.10279691

[28] Y. Zhang, B. Asulba, N. Schumacher, M. Sousa, P. Souto, L. Almeida, P. Santos, N. Martins, and J. Sousa. 2023. Implementing and Deploying an ML Pipeline for IoT Intrusion Detection with Node-RED. In *CPS-IoT Week Workshops '23: Cyber-Physical Systems and Internet of Things Week 2023* (San Antonio, TX, USA). https://doi.org/10.1145/3576914.3589807

[29] Marek Żyliński, Amir Nassibi, Ildar Rakhmatulin, Adil Malik, Christos Papavassiliou, and Danilo P. Mandic. 2024. Deployment of Artificial Intelligence Models on Edge Devices: A Tutorial Brief. *IEEE Transactions on Circuits and Systems II: Express Briefs* 71 (2024), 1738–1743. https://api.semanticscholar.org/CorpusID:265492906