

# Are Trees Really Green? A Detection Approach of IoT Malware Attacks

Silvia Lucia Sanna<sup>1</sup>[0009–0002–8269–9777], Diego Soi<sup>1</sup>[0009–0009–0092–9067],  
Davide Maiorca<sup>1</sup>[0000–0003–2640–4663], and Giorgio  
Giacinto<sup>1,2</sup>[0000–0002–5759–3017]

<sup>1</sup> Dip. Ingegneria Elettrica ed Elettronica,  
Università degli Studi di Cagliari, Cagliari, Italy

<sup>2</sup> CINI, Consorzio Interuniversitario Nazionale per l'Informatica, Roma, Italy  
{silvia.l.sanna, diego.soi, davide.maiorca, giorgio.giacinto}@unica.it

**Abstract.** Nowadays, the Internet of Things (IoT) is widely employed, and its usage is growing exponentially because it facilitates remote monitoring, predictive maintenance, and data-driven decision making, especially in the healthcare and industrial sectors. However, IoT devices remain vulnerable due to their resource constraints and difficulty in applying security patches. Consequently, various cybersecurity attacks are reported daily, such as Denial of Service, particularly in IoT-driven solutions.

Most attack detection methodologies are based on Machine Learning (ML) techniques, which can detect attack patterns. However, the focus is more on identification rather than considering the impact of ML algorithms on computational resources.

This paper proposes a *green* methodology to identify IoT malware networking attacks based on flow privacy-preserving statistical features. In particular, the hyperparameters of three tree-based models – Decision Trees, Random Forest and Extra-Trees – are optimized based on energy consumption and test-time performance in terms of Matthew's Correlation Coefficient.

Our results show that models maintain high performance and detection accuracy while consistently reducing power usage in terms of watt-hours (Wh). This suggests that on-premise ML-based Intrusion Detection Systems are suitable for IoT and other resource-constrained devices.

**Keywords:** Green ML, IoT, Malware Traffic, Network Analysis

## 1 Introduction

Nowadays, Internet of Things (IoT) devices and their interconnections are becoming exponentially important in everyday life, from industry [32] to houses, vehicles, and smart cities of interconnected systems [20]. Such devices have low capabilities in terms of energy and computational resources compared to desktop and server computers. They are often employed to measure specific data, i.e., *sensors*, and control mechanical systems or forces, i.e., *actuators*. With the

advent of Artificial Intelligence (AI), these devices are becoming more intelligent and capable of making decisions independently [2].

Over the years, IoT devices and interconnected networks have been found to be susceptible to various vulnerabilities [21], which have allowed several types of cybersecurity attacks, such as data exfiltration, Denial of Service (DoS), and its Distributed variant (DDoS). Most of the time, these attacks are perpetrated by malware specifically designed for IoT systems, which are generally equipped with limited operating systems (OS) based on the Linux kernel, such as *Mirai* [1], and *Torii* [30], and *Hide&Seek* [7]. Detecting the presence of malware in an IoT environment is crucial as a first step to counteract a large number of cyberattacks, given the growing importance of IoT devices, their security disadvantages [25], and the limited resources they are provided with.

In this work, we focused on three detection strategies based on network communications made by IoT devices, which normally communicate with remote servers for data exchange. When malware takes control over the expected operability, the packet transmissions change, and attack patterns can be identified to detect anomalous behavior. These systems, called Intrusion Detection Systems (IDS), are generally deployed on-premise in ad-hoc devices, i.e., firewalls or routers, which work as access points for the IoT devices, or they are host-based, running directly on the device under analysis. However, these systems are typically resource-consuming, especially if they rely on Machine Learning (ML) algorithms [12], which require significant computational power and memory, making them unsuitable for deployment on resource-constrained devices.

To the best of our knowledge, few works at the state of the art focus on *Green Machine Learning* for network security applications [16, 27], that is, optimizing the underlying ML model to find a balance between drained energy and performance. Indeed, a green ML strategy is necessary mainly for two aspects. First, maintaining high detection performance while reducing consumed energy allows the deployment of IDS on-premise directly in IoT devices. Second, the longer an attack remains undetected, the greater the energy consumption will be due to malware operation and legitimate activities the IoT device may perform. As a consequence, the operability lifecycle of the device diminishes, causing the so-called *e-waste* [22] of electronics. Therefore, reducing the size of detection algorithms and the consumed energy would increase the device's life. Additionally, reducing the overall energy consumed is fundamental to decreasing the equivalent carbon footprint, helping to combat climate change.

In this paper, we employed a dataset of common network cyberattacks in different IoT scenarios [26] (e.g., DoS and port-scanning) to identify post-mortem anomalies based on network features. We optimized the hyperparameters of three tree-based ML algorithms to minimize energy consumption during the testing phase while maximizing performance. We demonstrate that accuracy is not significantly affected by lower power usage. Our approach can be seen as an adaptive, energy-efficient IDS designed for network traffic detection.

To this end, we sketch two modes of operation:

- a) *non-green*, i.e., the training phase, in which the ML algorithm is optimized on a server, enabling fast learning and handling large training datasets, while still prioritizing power usage reduction during testing;
- b) *green* mode during runtime in resource-constrained devices, triggering alerts when an anomaly is detected as in the testing phase.

In this way, the algorithm is trained with large datasets to account for most learning patterns, while the algorithm running on-premise lowers power consumption, maintaining similar accuracy.

In summary, *i*) we developed a detection methodology based on networking post-mortem features, optimizing the ML algorithms for energy consumption in terms of  $\mu\text{Wh}$  and performance during the testing phase; *ii*) we selected only statistical features per flow without considering the body of the packets, i.e., a *privacy-preserving* approach; *iii*) we compared different ML algorithms to understand which is the most *green* during the testing phase; and *iv*) we defined the importance of false negative flows not detected by the system.

The rest of the paper is organized as follows. Section 2 reviews the literature on detecting malware networking communication and the energy consumption of ML systems, and Section 3 describes the dataset, features, and employed algorithms. Section 4 discusses the results of the classification with respect to the performance and energy consumption of the models. Finally, Section 5 discusses the limitations of the approach and future works.

## 2 Related Works

This section reviews recent works on current advancements in energy estimation with respect to AI technologies in Section 2.1, and the state-of-the-art regarding the detection of IoT malware through traffic patterns in Section 2.2.

### 2.1 Energy Consumption Measurement

Advancements in computer science technology powered by Cloud Computing and AI have substantially increased the demand for energy resources. This rise in energy consumption affects the feasibility of implementing cybersecurity solutions in battery-powered systems like IoT and mobile environments, while also contributing to  $\text{CO}_2$  emissions, exacerbating environmental challenges such as climate change. Thus, the reduction of software energy consumption is becoming an interesting topic for the research community [9, 34, 33]. A recent survey [13] reviewed two techniques for estimating the energy consumption of algorithms to better design software: *i*) *hardware-level* to compute the energy efficiency of hardware components (i.e., CPU, RAM, and I/O peripherals); *ii*) *software-level* through simulation or real-time estimation at the instruction level to trace the consumed energy by performance counter profiling or instruction-set simulation.

Among the most recent approaches, Budenny et al. proposed Eco2AI [6], a framework that measures energy consumption in terms of Joules or KWh,

focusing on CPU, GPU, and RAM real-time evaluation. `PyJoule`<sup>3</sup> employs the Intel RAPL (Running Average Power Limit) technology to estimate the power consumption of CPU, RAM and integrated GPU. Additionally, Antony et al. proposed `Carbon Tracker`, a tool to track the energy and carbon footprint of ML models. The authors evaluated the tool’s efficiency by comparing the estimations with the actual measurements done in monitoring 1 training epoch for two models, i.e., CNN and Autoencoders, with errors between 5% and 19%. Due to its efficiency and attested good results, in this work we based the optimization of detection algorithms on the measurements done by `Carbon Tracker` as explained in Section 3.

Other works, besides measuring the energy efficiency of neural network models and their training, suggest ways to reduce consumption. Tipp et al. [28] suggest, among other methods, reducing idle time when accessing memory to eliminate excess energy due to idle power drawn and reducing memory access by using specialized hardware to hold larger parameters in cache.

## 2.2 Detection of IoT Malware Traffic

The rapid evolution of IoT devices has attracted malware authors interested in exploiting security vulnerabilities through malicious software, whose aim is generally to gain unauthorized privileges in the network to which IoT devices are connected, like in the case of `Mirai` [1] and `SILEX` [23] attacks. For this reason, one of the main approaches to identify this kind of malicious activity is to leverage the network patterns they generate.

One of the first works was published by Bilge et al. [5] in 2012, whose main goal was to propose invariant network features without considering the application protocols due to differences in each client/server communication. In particular, they selected flow size-based, client access pattern-based features, and temporal ones to characterize the variability of client flow volume as a function of time. Recently, Davanian et al. [11] proposed a methodology to identify live C&C servers with zero-priori knowledge to separate the C2-bound traffic from other traffic accurately. Their methodology is based on a SYN-DATA-aware approach, depending on the number of SYN flags and the data exchanged. Moreover, they focus on a grammar-based representation of the traffic, considered as a dialog, to create a fingerprint-aware identification method. Barradas et al. [4] adapted the existing methodologies for C&C traffic with TLS 1.3 protocol, which improves the TCP handshake protocol. They employed features related to the packet sizes, discarding all timing features since they are affected by the distance between client and server, as well as by network conditions.

Other works, have addressed the multi-classification problem to identify several kinds of attacks in IoT networks, e.g., Denial of Service (DoS), and Port Scanning with several Machine Learning algorithms [18, 10] by considering both temporal and content-based features reaching accuracies in the order of 90%.

<sup>3</sup> <https://github.com/powerapi-ng/pyJoules>

Despite the importance of in-edge attack identification [17, 15], and its efficient resource management, few existing works addresses the usage of green machine learning algorithms for network security applications [16, 27]. The current proposed approaches (e.g., TinyML<sup>4</sup> techniques) reduce the size of learning algorithms to be suitable for IoT devices, selecting one method over the other without a real optimization step based on the power usage.

### 3 Methodology

This section introduces the proposed methodology to optimize ML models based on both energy consumption and performance, as depicted in Figure 1. Specifically, Section 3.1 describes the employed dataset, Section 3.2 discusses the extracted features, and Section 3.3 introduces the model training approach with the optimization strategy.

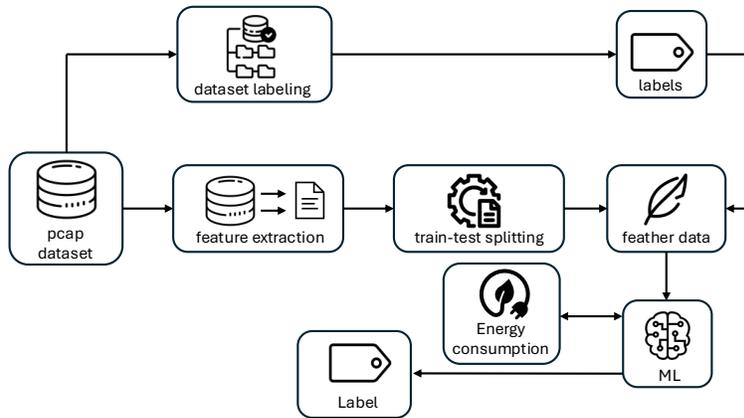


Fig. 1: Training workflow with the basic blocks of the methodology: dataset labeling, feature extraction, training, and energy-based optimization.

#### 3.1 Dataset

To evaluate the approach, we used the Aposemat IoT-23 dataset [26] by Garcia et al., a freely available labeled dataset of multiple PCAP network capture files. The dataset contains benign and malicious traffic involving a variety of IoT sources (e.g., smart hubs, smart lights, door lock devices) and malware.

<sup>4</sup> <https://github.com/mit-han-lab/tinyml>

The `NFStream`<sup>5</sup> tool was employed to analyze the PCAP files, obtain the label of the flows, and extract several meaningful statistical features, computed when a flow is closed, (i.e., utilizing the totality of a connection), since our approach is tailored to *post-mortem* traffic analysis. To identify each flow in the PCAP, we employed the five-tuple (source IP address, source IP port, destination IP address, destination IP port, timestamp) to match the identifiers of the original dataset in the file labels. The dataset was originally labeled by the authors Garcia et al. using the network traffic analyzer `Zeek`<sup>6</sup>. Moreover, to remove noise, we dropped the flows with multiple contrasting labels (e.g., a flow belonging both to a `Mirai` and `Kenjiro` botnet), or with no existing match in the `Zeek` results.

Table 1 shows the list of flow counts per class. Our processed dataset contains malicious traffic from seven malware tools: `Kenjiro`, `Mirai`, `Hakai`, `IRCBOT`, `Hajime`, `Hide&Seek`, and `Muhstick`, which represent the majority of malware found in real-world scenarios [31]. As noticeable, the majority of malware samples belong to the port-scan attack and the dataset is unbalanced in the malware/benign flows ratio. However, as it will be detailed in Section 4.4, we reduced it by removing this class of attack, making the dataset nearly balanced and obtaining better results.

CLASS	FAMILY	FLOWS
Malicious	IRCBOT – port scan	3627968
	Kenjiro – port scan	3525075
	Mirai – port scan	3236207
	Hajime – port scan	506947
	Hide and Seek – port scan	9558
	Muhstik – port scan	3671
	Mirai – C&C	559
	Hakai – C&C	103
	Kenjiro – DoS	776087
	Mirai – DoS	18344
Muhstik – DoS	298	
Benign	-	1532194

Table 1: Flow count grouped by class and attack types, i.e., port scan, C&C and DoS attacks.

<sup>5</sup> <https://www.nfstream.org/>

<sup>6</sup> <https://zeek.org/>

### 3.2 Extracted Features

NFStream was also employed to extract various statistical features, listed in Table 2 for each correctly labeled flow. These features were chosen since using statistics computed per flow in combination with Machine Learning techniques to identify attacks is a time-proven approach in the scientific literature [8, 3]. Specifically, we computed each feature for three representations: *i*) *bidirectional*, which includes packets exchanged in both directions of the communication, i.e., source and destination; *ii*) *source-to-destination*, where features are calculated solely on packets sent from the source to the destination; and *iii*) *destination-to-source*, which focuses on packets flowing in the reverse direction. We purposely omitted the IP addresses and ports from the list of features for training and testing our models since these values are either meaningless or easy to spoof and especially to train a more generic model adaptable to every situation, while preserving privacy.

FEATURE	UNIT
protocol	
IP version	
flow duration	
maximum packet inter-arrival time	
minimum packet inter-arrival time	ms
mean packet inter-arrival time	
standard deviation packet inter-arrival time	
transmitted bytes	
maximum packet size	
minimum packet size	bytes
mean packet size	
standard deviation packet size	
transmitted packets	packets
TCP packets with ACK set	
TCP packets with CWR set	
TCP packets with ECE set	
TCP packets with FIN set	
TCP packets with PSH set	packets
TCP packets with RST set	
TCP packets with SYN set	
TCP packets with URG set	

Table 2: Statistical features employed in our methodology. Each statistic has been computed for the bidirectional, source-to-destination, and destination-to-source communications.

Our statistics are based solely on packet timings, and they are computed only by analyzing the IP and TCP headers, which identify a flow in a communication. Moreover, the IP payload is not processed, and, therefore, this approach allows us to be encryption-agnostic with two main advantages. First, our methodology works equally well when the IP payload is encrypted (e.g., with a TLS or DTLS connection). Second, our approach safeguards the users' privacy since the content of the IP packets is never inspected.

Once we built our dataset, we randomly split it into a training and a test set following an 80-20 ratio.

### 3.3 Model training

In our experiments, we tested three of the most employed Machine Learning algorithms for network security applications [8, 3] using the well-known Python package `scikit-learn`<sup>7</sup>. In particular, the models are tree-based techniques: *i*) a *decision tree*, or single-tree [24], which is a hierarchical model that recursively splits data based on feature conditions to make predictions. Internal nodes represent decisions, and leaf nodes represent outcomes; *ii*) *random forest* [19] that is an ensemble of multiple decision trees that usually shows improved accuracy over a single tree by carefully deciding how to split the nodes; and *iii*) *extra-trees* (Extremely Randomized Trees) [14], which are also ensembles of trees, but they split the nodes randomly.

For each algorithm, we leveraged `optuna`<sup>8</sup>, a well-known hyperparameter optimization framework that helps to automate parameter search, to train four versions of each model given a function to optimize:

- a *default* model, that is, the model trained with the default hyperparameters of `scikit-learn`;
- a *max green* model, that is, the optimized model with the lowest energy consumption at testing time;
- a *max MCC* model, that is, the optimized model with the highest MCC (Matthew's Correlation Coefficient<sup>9</sup>);
- a *balanced* model, that is, the model obtained with a multi-objective optimization to maximize the MCC and minimize the energy consumption. Due to how `optuna` works, we might encounter multiple optimal models. As specified later in Section 4.3, we selected the best model as the one that is geometrically closest to the point (0, 1) in the Pareto front, where the first value is the power consumption and the second is the MCC. In other words, a model offering good discriminating capabilities and power saving without sacrificing too much of the two metrics.

<sup>7</sup> <https://scikit-learn.org/>

<sup>8</sup> <https://optuna.org/>

<sup>9</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews\\_corrcoef.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews_corrcoef.html)

To compute consumed energy, we employed **Carbon Tracker** which is, as outlined in Section 2.1, the best tool at the state of the art able to estimate the actual power usage.

Model optimization was performed based on energy consumption and performance during the testing phase. That is because the ultimate goal is to lower resources for the running algorithm on-premises while the training is performed on the server to account for large datasets. Additionally, due to class imbalance, model performance is computed with the Matthew’s Correlation Coefficient that considers all four values of the confusion matrix, i.e., True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

CPU	Intel® Core™ i9-11950H CPU @ 2.60GHz
RAM	32 GiB
OS	Debian GNU/Linux
kernel	6.2.10
Python	3.13.1
<code>scikit-learn</code>	1.6.1
<code>optuna</code>	4.2.0
<code>CarbonTracker</code>	2.0.1

Table 3: Specifics of our experimental setup.

## 4 Results

This Section discusses the results we obtained. In particular, Section 4.1 outlines the experimental setup we employed, Section 4.2 gives an overview of the obtained general results, Section 4.3 concerns the hyperparameter tuning results, and Section 4.4 examines the False Negative samples.

### 4.1 Experimental setup

The experiments were conducted on a machine equipped with the specifications in Table 3. In this preliminary work, we employed a server machine for both the training and testing phases. First, the server is used in the training for large-scale datasets, which need many resources in terms of memory (RAM), CPU/GPU usage, storage capabilities, and battery life that IoT devices do not support. We preferred to use the server even in the testing phase to ensure reproducibility, precise power measurement, and full control of the experimental setup. As there is a wide variety of different IoT devices and some of them have proprietary systems, achieving large reproducibility, adaptability, and comparison of the results on different systems is difficult because of the different available environments.

TYPE	VERSION	HYPERPARAMETER		
		MAX_DEPTH	MIN_LEAF	MIN_SPLIT
single-tree	default	$\infty$	1	2
	max green	1	3	9
	max MCC	14	5	29
	balanced	13	5	13

(a) Hyperparameters for single-tree classifiers.

TYPE	VERSION	HYPERPARAMETER				
		MAX_DEPTH	MIN_LEAF	MIN_SPLIT	MAX_FEAT	ESTIM.
random forest	default	$\infty$	1	2	sqrt	100
	max green	71	7	29	14	10
	max MCC	11	6	17	11	133
	balanced	17	6	20	7	18

(b) Hyperparameters for random forest classifiers.

TYPE	VERSION	HYPERPARAMETER				
		MAX_DEPTH	MIN_LEAF	MIN_SPLIT	MAX_FEAT	ESTIM.
extra trees	default	$\infty$	1	2	sqrt	100
	max green	169	6	18	6	10
	max MCC	18	2	20	23	205
	balanced	14	2	18	24	204

(c) Hyperparameters for extra-trees classifiers.

Table 4: Hyperparameter chosen by `optuna` for each model.

In fact, as mentioned in Section 4.2, the consumed energy remains low (in the order of  $\mu\text{Wh}$ ), which is in line with the constrained resources of IoT devices [2]. Therefore, while the experiments were conducted on a server, the methodology remains compatible with resource-constrained environments.

## 4.2 Overview

Tables 4 and 5 report the selected hyperparameters by `optuna` for each of the trained models, and the average energy consumption per testing sample and model performance in terms of Matthews' Correlation Coefficient (MCC), balanced accuracy, and F1-score to account for class imbalance in the dataset. These scores are chosen to consider both the dataset imbalance and the need to keep false negatives low to avoid malicious flows going undetected.

TYPE	VERSION	$\mu$ Wh	MCC	B. ACC.	F1
single-tree	default	19.35	0.52	94.70	75.46
	max green	6.50	0.23	89.05	6.42
	max MCC	8.13	0.60	95.35	76.11
	balanced	7.93	0.60	95.32	72.10
random forest	default	299.83	0.52	94.68	75.76
	max green	22.72	0.58	95.24	76.70
	max MCC	124.90	0.61	95.33	73.85
	balanced	28.70	0.61	95.33	74.20
extra-trees	default	284.45	0.526	94.69	75.64
	max green	22.04	0.57	95.09	72.26
	max MCC	213.38	0.61	95.33	74.38
	balanced	49.68	0.60	95.24	74.08

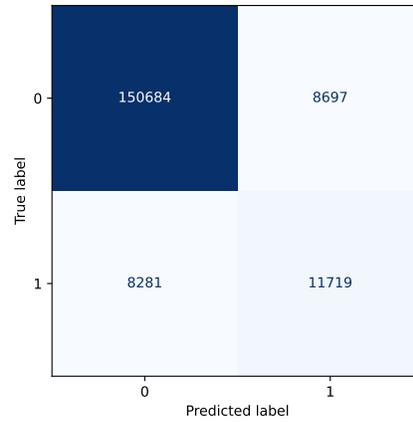
Table 5: Performance statistics of our classifiers. In particular, for each model, performance is shown in terms of average  $\mu$ Wh per test sample, Matthews’ Correlation Coefficient (MCC), Balanced Accuracy, and F1-score.

Interestingly, the default models offer strong discriminating power, reaching about 99% balanced accuracy and 76% F1-score. However, they are always the most energy-hungry, consuming about 13 times more  $\mu$ Wh than their max green counterparts. This is consistent with the default hyperparameters (see Table 4) in `scikit-learn`, which were chosen to offer good performance in multiple scenarios, completely ignoring power consumption.

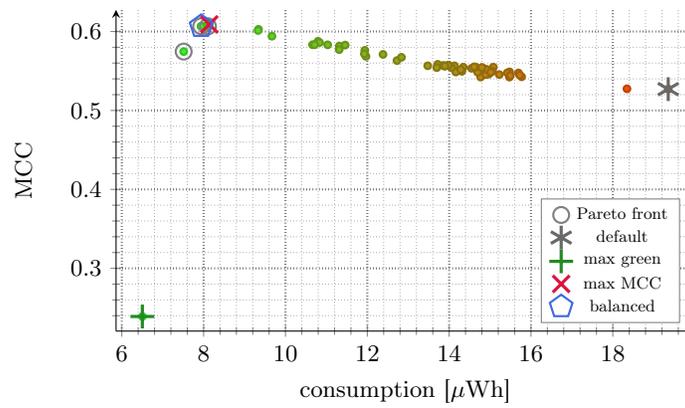
As expected, the max green models are the most eco-friendly. However, this version of the single-tree has poor performance with 6% of F1-score and 0.239 of MCC. This suggests that the green model has a weak correlation between its prediction and the label class. These relatively poor results may be caused by `optuna` selecting a max depth of 1 (see Table 4a), meaning that the model makes a single split based on one feature, oversimplifying the classification. Instead, the other versions of the single-tree seem to be the most efficient and high-performing, even with respect to the other models, i.e., random-forest and extra-trees, which still offer good performance statistics but are highly resource-demanding due to the ensemble nature of these classifiers, which aggregate multiple weak learners, requiring more CPU and memory.

### 4.3 Optimization of the balanced models

This section discusses the hyperparameter tuning process of the balanced version of the selected models. For each optimization, we asked `optuna` to perform 64 iterations, i.e., the best number of iterations to achieve good optimization. The optimizations aimed to maximize the MCC and minimize the energy footprint during the inference phase. In particular, we refer only to the single-tree model,



(a) Simplified Confusion Matrix of the balanced single-tree model.



(b) Pareto front for the single-tree models. The x-axis shows the mean energy consumption in terms of  $\mu\text{Wh}$ , and the y-axis is the MCC score.

Fig. 2: Confusion Matrix and Pareto front for the single-tree models showing classification accuracy and the performance in relation to the consumed energy.

which was found to be the most efficient and to have the highest performance as outlined in Section 4.2. Similar considerations can be made for the other models.

The Pareto front in Figure 2b helps in understanding how `optuna` selects the best models. It shows all the single-tree models tested by `optuna` in a scatter plot, where the points represent the Pareto front, the x-axis is the mean energy consumption per sample, and the y-axis represents the performance in terms of MCC. A solution is considered Pareto-optimal if no other configuration performs better in both objectives simultaneously [29]. Points on the front are non-dominated, meaning improving one metric would negatively impact the other.

For example, increasing the MCC beyond a certain point may require a model that consumes significantly more energy, while reducing energy usage might come at the cost of lower classification accuracy. This provides a valuable decision boundary, allowing model selection based on application-specific priorities, such as maximizing accuracy, minimizing energy usage, or achieving a balanced compromise.

The default model (black star) presents high energy consumption ( $\sim 19 \mu\text{Wh}$ ) with moderate MCC ( $\sim 0.53$ ) and, as noticeable even from Table 5, it is not efficient but is a reference point for suboptimal optimizations of the model. The max green (green cross) has the lowest consumption ( $\sim 7 \mu\text{Wh}$ ), but low MCC ( $\sim 0.4$ ), i.e., it is optimized only for energy consumption, suitable only if energy minimization is the priority, and accuracy is less critical. Conversely, the max MCC (red cross) is appropriate when maximizing the performance is critical, even though the consumed energy is not as low as for the green model ( $\sim 8 \mu\text{Wh}$ ). Finally, the balanced model (blue pentagon) is similar to the max MCC version with similar consumed energy and performance. Indeed, they both remain eco-friendly, maintaining the generalization capability. This means that both max MCC and balanced variants are suitable for running on an IoT device.

#### 4.4 Error analysis

As outlined in Sections 4.2 and 4.3, single-tree model variants are the most efficient both in terms of performance and energy consumption. In particular, the balanced model, even though similar to the max MCC version, requires fewer resources and achieves comparable MCC. Figure 2a shows the model confusion matrix, depicting that legitimate samples are well recognized, while it fails to clearly recognize a good portion of malicious flows—over 40% of actual true positives are misclassified. Therefore, even though MCC is good, i.e., greater than 0, it is not good enough to be comparable with other works at the state-of-the-art [11] and requires attention due to the criticality of the application.

Analyzing the dataset and the corresponding features, we found that the models failed to recognize port-scan attacks whose flows are similar to legitimate ones. Therefore, we conducted a study of methodology performance while removing port-scan attacks from the dataset in Table 1, training only the single-tree model, which was the best performing of the three selected algorithms. We

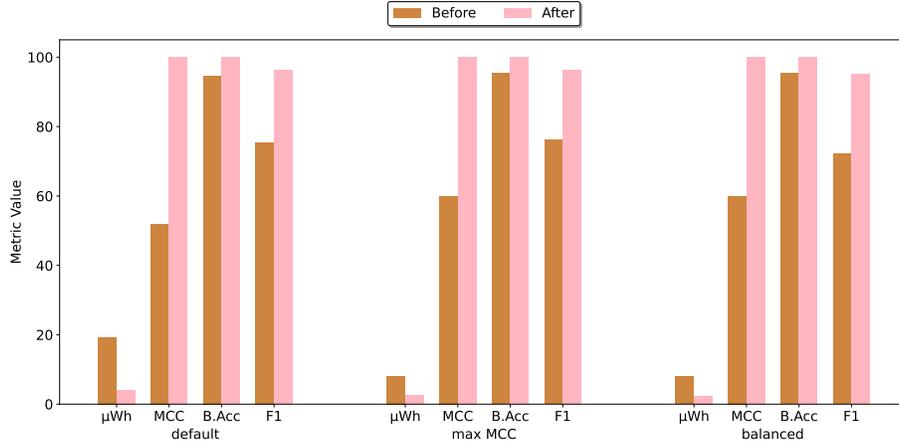


Fig. 3: Comparison between performance of the optimized single-tree models (i.e., default, max MCC, and balanced) before (left bars) and after (right bars) port-scanning attack removal. MCC is multiplied by 100 to be suitable for the graph.

did not optimize the hyperparameters to achieve optimal energy consumption, as we expected it to have low accuracy.

Figure 3 shows a comparison of single-tree model performance before and after the removal of port-scan attacks. As noticeable, MCC, Balanced Accuracy and F1 score increased after removing port-scan attacks while the average consumed  $\mu\text{Wh}$  reduced. The latter is expected because the less are the flows, the less complex the model is and therefore less resources are employed.

As before, we analyze the Pareto front in Figure 4. It includes many points tightly clustered around low consumption ( $\sim 2.3\text{--}2.6 \mu\text{Wh}$ ) with near-perfect MCC, suggesting that it is possible to reduce energy without compromising the performance. For example, the max MCC (red cross) offers the absolute best MCC performance (1.0) but at a slightly higher energy cost, while the balanced model (blue pentagon) lies on the Pareto front, delivering strong performance (0.995) with modest energy use ( $2.35 \mu\text{Wh}$ ). On the contrary, the default model (black star), despite achieving a high MCC (0.9997), is inefficient because it consumes significantly more energy than necessary, i.e.,  $\sim 50\%$  more than the other two versions, for comparable accuracy, as discussed in Sections 4.2 and 4.3.

## 5 Conclusion

In this work, we tested the energy efficiency of tree-based Machine Learning algorithms trained to detect malicious network traffic generated by common IoT malware. The methodology is based on flow statistical features to preserve the privacy of legitimate communications. Additionally, we developed an optimization

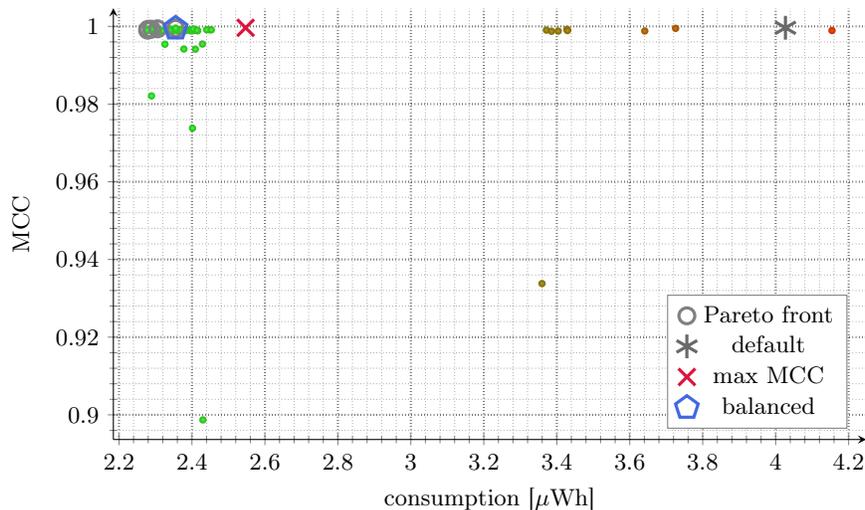


Fig. 4: Pareto front for the tree models after removing port-scanning flows.

strategy with `optuna` and `Carbon Tracker` based on testing phase performance, considering both power consumption in Wh and MCC, while the training stage does not take power limitations into consideration. The reason is twofold. First, the trained detection algorithms can run on-premises, ideally on constrained IoT devices. Second, reducing the energy impact of the testing stage counteracts energy waste, increases the device operability lifecycle, and reduces the carbon footprint.

We tested the methodology on a dataset consisting of three IoT attacks, reaching interesting results. Indeed, the models maintain high performance while keeping low energy consumption. The balanced version of the models, i.e., models trained to balance both MCC and  $\mu\text{Wh}$ , attained about 0.60 MCC and a reduction of 60 – 90% in consumed resources compared to the models trained with the default hyperparameters. These results suggest that ML-based IDS systems are suitable for running on on-premise devices. Additionally, we studied the model errors and found that the dataset has biases with respect to port-scanning attacks, which have similar features to legitimate traffic flows.

However, the proposed approach still has some limitations. It is tested on only one dataset, limiting generalizability with respect to different network topologies and attacks. Additionally, the experimental setup lacks a constrained device to run the trained model to compute energy efficiency. Indeed, the optimization of the hyperparameters with respect to the testing phase results is done on the same server where the models are trained.

In the future, we plan to improve our methodology with live analysis to test energy efficiency and inspect incoming network streaming. Additionally, we will test the approach with Deep Learning algorithms, which are inherently more energy-demanding than the tree-based algorithms we selected.

## References

1. What is the mirai botnet? <https://www.cloudflare.com/it-it/learning/ddos/glossary/mirai-botnet/>, 2025.
2. Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.
3. Ofek Bader, Adi Lichy, Chen Hajaj, Ran Dubin, and Amit Dvir. Maldist: From encrypted traffic classification to malware traffic detection and classification. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, pages 527–533, 2022.
4. Diogo Barradas, Carlos Novo, Bernardo Portela, Sofia Romeiro, and Nuno Santos. Extending c2 traffic detection methodologies: From tls 1.2 to tls 1.3-enabled malware. In *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses, RAID '24*, New York, NY, USA, 2024. Association for Computing Machinery.
5. Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC '12*, page 129–138, New York, NY, USA, 2012. Association for Computing Machinery.
6. Semen Budenny, Vladimir Lazarev, Nikita Zakharenko, Alexey Korovin, Olga Plosskaya, Denis Dimitrov, Vladimir Arkhipkin, Ivan Oseledets, Ivan Barsola, Ilya Egorov, Aleksandra Kosterina, and Leonid Zhukov. eco2ai: Carbon emissions tracking of machine learning models as the first step towards sustainable ai. *Doklady Mathematics*, 2022.
7. Trend Business. 'hide 'n seek' botnet uses peer-to-peer infrastructure to compromise iot devices. <https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/-hide-n-seek-botnet-uses-peer-to-peer-infrastructure-to-compromise-iot-devices>, 2018.
8. Daniele Canavese, Leonardo Regano, Cataldo Basile, Gabriele Ciravegna, and Antonio Lioy. Encryption-agnostic classifiers of traffic originators and their application to anomaly detection. *Computers & Electrical Engineering*, 97:107621, 2022.
9. Antonio Candelieri, Andrea Ponti, and Francesco Archetti. Fair and green hyperparameter optimization via multi-objective and multiple information source bayesian optimization. *Machine Learning*, 113, 2024.
10. Andrew Churcher, Rehmat Ullah, Jawad Ahmad, Sadaqat ur Rehman, Fawad Masood, Mandar Gogate, Fehaid Alqahtani, Boubakr Nour, and William J. Buchanan. An experimental analysis of attack classification using machine learning in iot networks. *Sensors*, 21(2), 2021.
11. Ali Davanian, Michail Faloutsos, and Martina Lindorfer. C2miner: Tricking iot malware into revealing live command & control servers. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, ASIA CCS '24*, New York, NY, USA, 2024. Association for Computing Machinery.
12. Eva García-Martín, Niklas Lavesson, Håkan Grahm, Emiliano Casalicchio, and Veselka Boeva. How to measure energy consumption in machine learning algorithms. In *ECML PKDD 2018 Workshops*, pages 243–255, Cham, 2019. Springer International Publishing.
13. Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahm. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, 2019.

14. Pierre Geurst, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63, 2006.
15. Truong Thu Huong, Ta Phuong Bac, Dao M. Long, Bui D. Thang, Nguyen T. Binh, Tran D. Luong, and Tran Kim Phuc. Lockedge: Low-complexity cyberattack detection in iot edge computing. *IEEE Access*, 9:29696–29710, 2021.
16. Iacovos Ioannou, Prabagarane Nagaradjane, Pelin Angin, Palaniappan Balasubramanian, Karthick Jeyagopal Kavitha, Palani Murugan, and Vasos Vassiliou. Gemlids-miot: A green effective machine learning intrusion detection system based on federated learning for medical iot network security hardening. *Computer Communications*, 218, 2024.
17. Yizhen Jia, Fangtian Zhong, Arwa Alrawais, Bei Gong, and Xiuzhen Cheng. Flow-guard: An intelligent edge defense mechanism against iot ddos attacks. *IEEE Internet of Things Journal*, 7(10):9552–9562, 2020.
18. Rakesh Kumar, Mayank Swarnkar, Gaurav Singal, and Neeraj Kumar. Iot network traffic classification using machine learning algorithms: An experimental analysis. *IEEE Internet of Things Journal*, 9(2):989–1008, 2022.
19. Yanli Liu, Yourong Wang, and Jian Zhang. New machine learning algorithm: Random forest. In Baoxiang Liu, Maode Ma, and Jincai Chang, editors, *Information Computing and Applications*, pages 246–252, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
20. Keertikumar M., Shubham M., and R.M. Banakar. Evolution of iot in smart vehicles: An overview. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 804–809, 2015.
21. Francesca Meneghello, Matteo Calore, Daniel Zucchetto, Michele Polese, and Andrea Zanella. Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices. *IEEE Internet of Things Journal*, 6(5):8182–8201, 2019.
22. Fathi Batoul Modarress, Alexander Ansari, and Ansari Al. Threats of internet-of-thing on environmental sustainability by e-waste. *Sustainability*, 2022.
23. Basem Ibrahim Mukhtar, Mahmoud Said Elsayed, Anca D. Jurcut, and Marianne A. Azer. Iot vulnerabilities and attacks: Silex malware case study. *Symmetry*, 15(11), 2023.
24. Arundhati Navada, Aamir Nizam Ansari, Siddharth Patil, and Balwant A. Sonkamble. Overview of use of decision tree algorithms in machine learning. In *2011 IEEE Control and System Graduate Research Colloquium*, pages 37–42, 2011.
25. Eryk Schiller, Andy Aidoo, Jara Fuhrer, Jonathan Stahl, Michael Ziörjen, and Burkhard Stiller. Landscape of iot security. *Computer Science Review*, 44:100467, 2022.
26. Maria Jose Erquiaga Sebastian Garcia, Agustin Parmisano. Iot-23: A labeled dataset with malicious and benign iot network traffic.
27. Nazli Tekin, Abbas Acar, Ahmet Aris, A. Selcuk Uluagac, and Vehbi Cagri Gungor. Energy consumption of on-device machine learning models for iot intrusion detection. *Internet of Things*, 21:100670, 2023.
28. Charles Edison Tripp, Jordan Perr-Sauer, Jamil Gafur, Amabarish Nag, Avi Purkayastha, Sagi Zisman, and Erik A. Bensen. Measuring the energy consumption and efficiency of deep neural networks: An empirical analysis and design recommendations, 2024.
29. Tea Tušar and Bogdan Filipič. Visualization of pareto front approximations in evolutionary multiobjective optimization: A critical review and the prosection method. *IEEE Transactions on Evolutionary Computation*, 19(2):225–245, 2015.

30. Jai Vijayan. 'torii' breaks new ground for iot malware. <https://www.darkreading.com/cyberattacks-data-breaches/-torii-breaks-new-ground-for-iot-malware>, 2018.
31. Huanran Wang, Weizhe Zhang, Hui He, Peng Liu, Daniel Xiapu Luo, Yang Liu, Jiawei Jiang, Yan Li, Xing Zhang, Wenmao Liu, Runzi Zhang, and Xing Lan. An evolutionary study of iot malware. *IEEE Internet of Things Journal*, 8(20):15422–15440, 2021.
32. Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014.
33. Tim Yarally, Lus Cruz, Daniel Feitosa, June Sallou, and Arie van Deursen. Uncovering energy-efficient practices in deep learning training: Preliminary steps towards green ai. In *2023 IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN)*, pages 25–36, 2023.
34. André M. Yokoyama, Mariza Ferro, and Bruno Schulze. A multi-objective hyperparameter optimization for machine learning using genetic algorithms: A green ai centric approach. In Ana Cristina Bicharra Garcia, Mariza Ferro, and Julio Cesar Rodríguez Ribón, editors, *Advances in Artificial Intelligence – IBERAMIA 2022*. Springer International Publishing, 2022.