Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

Energy consumption of on-device machine learning models for IoT intrusion detection

Nazli Tekin [a,b,*], Abbas Acar [a], Ahmet Aris [a], A. Selcuk Uluagac [a], Vehbi Cagri Gungor [c]

[a] Cyber-Physical Systems Security Lab, Department of Electrical and Computer Engineering, Florida International University, Miami, FL 33174, USA

[b] Department of Software Engineering, Erciyes University, 38280, Kayseri, Turkey

[c] Department of Computer Engineering, Abdullah Gul University, 38080 Kayseri, Turkey

ARTICLE INFO

Keywords:
On-device machine learning
Energy consumption
Intrusion detection
Smart home
IoT

ABSTRACT

Recently, Smart Home Systems (SHSs) have gained enormous popularity with the rapid development of the Internet of Things (IoT) technologies. Besides offering many tangible benefits, SHSs are vulnerable to attacks that lead to security and privacy concerns for SHS users. Machine learning (ML)-based Intrusion Detection Systems (IDS) are proposed to address such concerns. Conventionally, ML models are trained and tested on computationally powerful platforms such as cloud services. Nevertheless, the data shared with the cloud is vulnerable to privacy attacks and causes latency, which decreases the performance of real-time applications like intrusion detection systems. Therefore, on-device ML models, in which the user data is kept locally, have emerged as promising solutions to ensure the security and privacy of the data for real-time applications. However, performing ML tasks requires high energy consumption. To the best of our knowledge, no study has been conducted to analyze the energy consumption of ML-based IDS. Therefore, in this paper, we perform a comparative analysis of on-device ML algorithms in terms of energy consumption for IoT intrusion detection applications. For a thorough analysis, we study the training and inference phases separately. For training, we compare the cloud computing-based ML, edge computing-based ML, and IoT device-based ML approaches. For the inference, we evaluate the TinyML approach to run the ML algorithms on tiny IoT devices such as Micro Controller Units (MCUs). Comparative performance evaluations show that deploying the Decision Tree (DT) algorithm on-device gives better results in terms of training time, inference time, and power consumption.

1. Introduction

With the proliferation of the Internet of Things (IoT) technologies, Smart Home Systems (SHSs) have gained tremendous popularity. SHSs enable autonomous activities. For example, a smart thermostat that automatically regulates room temperature or a smart light that automatically turns on during a detected motion provide convenience to the users while providing energy savings [1]. Users also benefit from smart cameras and smart door locks with motion sensors to enhance their home security. In addition, smart health monitoring systems assist patients who need constant supervision [2].

Albeit many advantages, SHSs suffer from various cyber attacks such as Denial of Service (DoS) or botnet attacks that may lead to security and privacy concerns [3]. Such attacks can cause substantial damage to IoT services in SHSs. Additionally, SHSs

* Corresponding author at: Department of Software Engineering, Erciyes University, 38280, Kayseri, Turkey.
E-mail addresses: nazlitekin@erciyes.edu.tr (N. Tekin), aacar001@fiu.edu (A. Acar), aaris@fiu.edu (A. Aris), suluagac@fiu.edu (A.S. Uluagac), cagri.gungor@agu.edu.tr (V.C. Gungor).

https://doi.org/10.1016/j.iot.2022.100670
Received 7 November 2022; Received in revised form 8 December 2022; Accepted 11 December 2022

include several IoT devices that periodically collect sensitive information such as user credentials, user behavior/preferences and video records that can be manipulated/stolen/used for destructive activities by attackers [4]. Malicious activities may include system disablement or personal data breach [5]. Therefore, intrusion detection systems for SHSs have become imperative to provide security and privacy for the IoT devices deployed in SHSs and their users. Machine Learning (ML)-based intrusion detection is a promising solution to address the security and privacy concern of SHSs.

The training phase of such an ML-based intrusion detection system can be deployed in three different ways. First, conventionally, ML-based IoT intrusion detection systems are deployed with the assistance of the cloud services, in which the ML models are trained in the cloud due to its unlimited resources and massive storage capabilities. After the training, either the final model parameters or inference results are transmitted back to the device for the intrusion detection application running on the device. This approach has the advantage of cloud capabilities, but it also has drawbacks such as security and privacy concerns as well as long network delays and high energy consumption due to the data transmission. Second, edge computing is an alternative to cloud computing, in which the data is kept local and processed at an edge device [6]. Accordingly, performing ML on edge devices close to the data source or IoT device itself reduces storage usage in the cloud, high network bandwidth requirement, and latency. The latency drastically decreases as there is no data exchange between IoT devices and the cloud. However, the edge computing-based ML approach does not provide the highly configurable hardware opportunities and unlimited resources provided by cloud computing-based ML. Third, recent advancements in IoT devices have enabled bringing ML tasks on-device. On-device ML solutions have been proposed in the literature to meet real-time application needs [7]. It usually refers to learning/training on the IoT end devices. Additionally, similar to edge computing, running ML models on an IoT end device provides data privacy since the data is not shared with other entities. Despite its advantages, ML tasks require huge computation power and massive data storage.

In addition to different training approaches, the inference phase of an ML-based IoT intrusion detection system can also be deployed in two different ways. In the first method, the pre-trained model, which is trained using any of the methods above, can be transmitted to the IoT end device without any conversion. This method can only be applied to programmable devices. In the second method, model conversion techniques like TinyML [8] can be applied to convert the ML model into the C code that can run on resource-constrained IoT devices. Despite different approaches for distributing the computation and resource usage among the application components, the energy consumption of ML algorithms remains a critical constraint for resource-constrained IoT devices. On the other hand, energy consumption can also be critical for cloud services, as the increase in the dataset size can cause an exponential increase in the energy consumed by the cloud service. Moreover, one can use a less energy-consuming algorithm if the same accuracy can be achieved. An optimal on-device ML-based solution for IDS should achieve better accuracy as well as energy efficiency. Because of these, a comparative evaluation of different on-device ML approaches for IoT intrusion detection applications in terms of energy consumption is required.

To this end, in this work, we evaluate the energy consumption of on-device ML learning algorithms, in which we compare three different training and two different inference approaches. We measure energy consumption via both execution time and power monitoring tool. Our results reveal some interesting findings: For example, we observed that training the Naive Bayes (NB) algorithm requires much less time than all other algorithms in all computing platforms, thereby consuming less energy. However, the accuracy performance of NB is lower than others. On the other hand, Decision Tree (DT) and k-Nearest Neighbor (k-NN) training also require relatively less time while providing the best accuracy together with Artificial Neural Network (ANN) and Random Forest (RF) in all computing platforms. Second, we found that while the cloud provides a significant decrease in execution time of the training for algorithms that support multiple cores, such as RF, it does not have the same impact on the other algorithms. Third, we found that a clustering algorithm k-NN spends much longer time in the inference than the training, so the inference in an IoT device would not be ideal for the computation even the training phase was performed on the cloud or edge. Finally, our energy consumption analysis via power monitoring tool verified that the DT algorithm provide the same accuracy while consuming much less power than other algorithms.

Contributions: Our contributions are listed as follows:

- We investigate the energy consumption of on-device ML models for IoT intrusion detection applications that can be deployed in SHSs. Particularly, for training, we compare cloud computing-based ML, edge computing-based ML, and IoT device-based ML approaches. On the other hand, we compare conventional and TinyML approaches for the inference phase.
- We implement each approach (three training + two inference approaches) and empirically evaluate the results.
- We show two different techniques to evaluate the energy consumption of ML algorithms.
- Our experiments revealed interesting findings. Our findings can be used as a guide while deploying a real-world ML-based IoT intrusion detection system.

Organization: The rest of the paper is organized as follows. Section 2 presents a motivational example. Section 3 provides background information on ML algorithms. Section 4 defines the on-device ML approaches for IDS. Section 5 presents our experiment methodology. Section 6 presents our results. Section 7 provides a summary of the previous works. Finally, Section 8 concludes the paper.

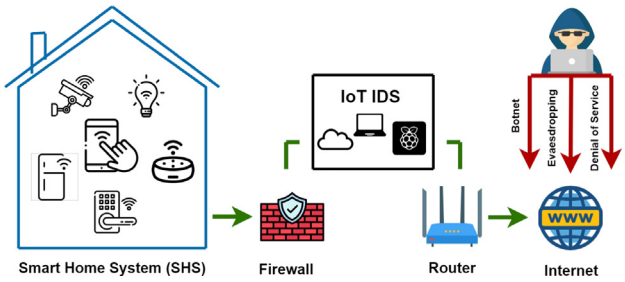


Fig. 1. Example Intrusion Detection System (IDS) for SHSs.

2. Motivational example

Real-time IoT Intrusion Detection System: SHSs consist of various IoT devices that may be exposed to a wide range of attacks such as denial of service (DoS) or botnet attacks. An example of such an attack is the Mirai botnet, which infected over 600k IoT devices [9]. These attacks generally are used to target high-profile networks, but they can also be used to target local IoT users. For example, ransomware [10] or cryptojacking [11] are some examples of IoT users/devices being the victim of such attacks. On the other hand, the IoT attack surface also includes the network attacks such as eavesdropping or spoofing [12]. With these attacks, the attacker can learn the daily routine of the IoT device user and can cause attacks like physical harm, burglary, or hardware damage. Moreover, the attackers can also target low-security IoT devices in SHSs. For example, these attacks can disable the smart lock or camera if they access the user's smartphone apps [13]. Therefore, real-time IoT intrusion detection is very critical. Fig. 1 illustrates an example of IDS. On-device ML models enable real-time IDS. However, running ML models on IoT devices is challenging due to their limited computation power and memory. On the other hand, most IoT devices are battery-powered, and the batteries deplete while performing energy-intensive ML algorithms. Therefore, in this paper, we analyze the on-device ML-based IDS for SHSs in terms of energy consumption.

3. Background

An ML algorithm consists of two phases: training and inference. Training refers to building a model using ML algorithms and training datasets, whereas inference refers to predicting a new data instance using a pre-trained ML model. This section gives background information about the ML algorithms we used in this study.

Logistic Regression: Logistic Regression (LR) is a widely used supervised ML technique for classification predictions. LR models use the sigmoid function to provide binary classification. The probability of binary classification output based on the sigmoid function is given by

$$P(x) = 1 + \frac{1}{e^{-(ax+b)}}, \tag{1}$$

where x is an input, and a and b are the slope and intercept of the learning model, respectively.

k-Nearest Neighbor: K-Nearest Neighbor (k-NN) is a non-parametric supervised ML algorithm for both classification and regression. The k-NN algorithm does not generate a mathematical model for training. Instead, it calculates the distance between input data and instances of training data to identify the k number of nearest neighbors. Each training data votes for a classification label and the input data are classified with the most voted label among its nearest neighbors.

Decision Tree: Decision Tree (DT) is another non-parametric supervised ML algorithm used for both classification and regression. DT consists of internal nodes that indicate feature data, branches that indicate decision rules, and leaf nodes that indicate predicted value. The algorithm creates a set of if-then-else decision rules to perform classification. There are various DT algorithms such as Iterative Dichotomiser 3 (ID3), C4.5, C5.0, and Classification and Regression Trees (CART). In this paper, we use CART from the scikit-learn library [14] to build a tree for anomaly detection.

Random Forest: The Random Forest (RF) algorithm performs classifications by aggregating many decision trees to reduce overfitting. The decision trees are trained in parallel by using various random subsets of features in the dataset. In the end, the prediction is performed according to the majority decisions of the trees.

Naive Bayes: Naive Bayes (NB) is a supervised ML model based on the Bayes theorem [15]. The model calculates the probability that a new data instance belongs to a particular class and performs classification by selecting the highest probability. Therefore, the classification model is given by [15]:

$$C(d) = \arg\max p(c) \prod_{i=1}^{n} p(d_i|c), \tag{2}$$

where d is the instance data, d_i is the instance attribute, n is the number of attributes, and c is the class type. Gaussian NB is the type of NB that estimates the mean and standard deviation of the data, unlike the other functions that calculate the data distribution.

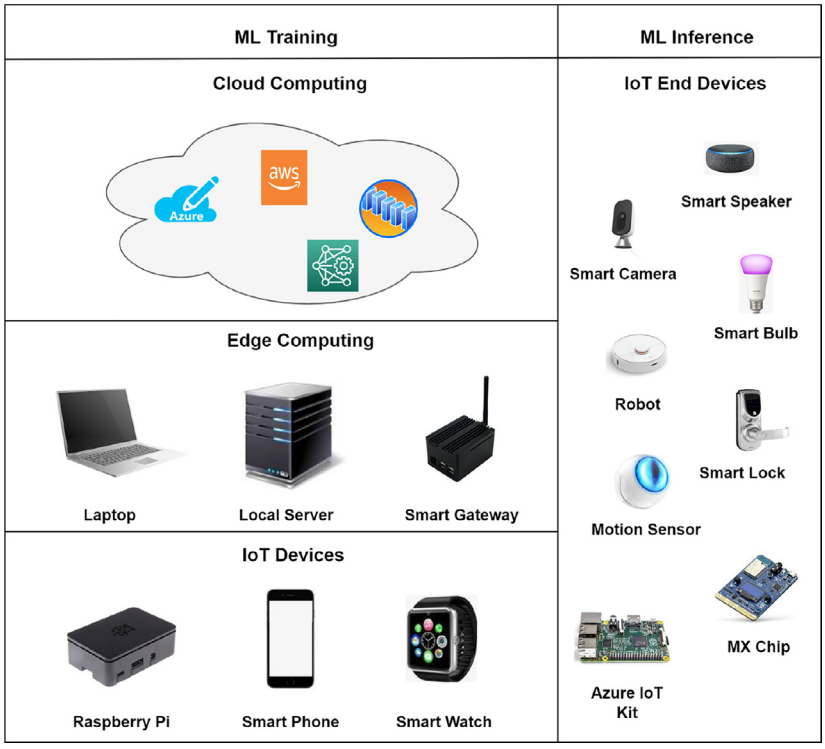


Fig. 2. Architecture of on-device ML deployment models.

Artificial Neural Networks: An artificial Neural Network (ANN) consists of input layers, one or more hidden layers, and an output layer. Input layers transfer the information through hidden layers that adjust and process the information. As the information goes through hidden layers, the network learns about the information and provides data to the output layer. Since Multilayer Perception (MLP) is the most common type of ANN, we use an MLP classifier from the scikit-learn library [14] in this study.

4. On-device machine learning deployment approaches

This section explains the different approaches that can be used to deploy on-device ML models for training and inference. The training phase of the ML algorithm can be performed on the cloud, edge device, or IoT device. Similarly, the inference phase can be performed on-device by utilizing two different methods that depend on the programmability of the end device. Provided that the end device allows running ML algorithm on the device (e.g., Raspberry Pi), the model can be directly deployed on the end device to perform inference. However, if the device is not programmable (e.g., Azure IoT Kit), a model conversion technique such as TinyML can be applied to transform the model into a machine code that can run on the end device. In the following sections, we explain the details of these approaches.

4.1. Training approaches

As shown in Fig. 2, the training phase of an ML-based intrusion detection application can be performed on the cloud, edge devices, or IoT devices. Then, the pre-trained model can be shared with the end device for on-device inference.

4.1.1. Cloud computing-based machine learning

Cloud computing is a service that provides computing resources such as memory, servers, databases, and networks through the Internet. Cloud services can be categorized into three main groups: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Basically, SaaS is the software delivery method for online access instead of installing the software. PaaS delivers platform resources such as development frameworks and operating system support. IaaS offers resources such as processing, storage, and network.

Cloud computing has ample storage and high processing power capabilities that can meet the increased computational needs of ML tasks. Big technology companies offer cloud-delivering services to support the high storage and computational power demand of ML-based IoT applications. For instance, Amazon Web Services (AWS), Google Cloud, and Microsoft Azure are major cloud computing platforms for performing ML models on the cloud.

4.1.2. Edge computing-based machine learning

Edge computing is an emerging approach that brings computation and data storage closer to IoT devices and sensors (i.e., data source) [6]. In addition to data processing and storage, the edge can provide delivery service between the cloud and the end-user. Generally, an edge device is defined as a computing platform with limited resources. It can be a powerful local server, computer, or even a small IoT device such as Raspberry Pi or a smartphone. Distributed edge devices such as gateways, PCs, IoT hubs, and smartphones provide data processing and storage services for a group of IoT devices. The advantages include supporting delay-sensitive applications by performing computation on edge devices close to the data generated by many IoT devices. Additionally, unlike cloud computing, edge computing does not require high network bandwidth to transmit a vast amount of data. Moreover, edge computing-based ML prevents data leakage by keeping the data locally. On the other hand, it reduces the energy cost of data transmissions to the cloud. Therefore, edge computing has the potential to tackle latency, high bandwidth requirements, and privacy challenges.

4.1.3. IoT device-based machine learning

IoT devices are resource-limited, so they have low computational power and memory. Generally, commercial-off-the-shelf IoT devices are not programmable. Therefore, performing ML algorithms on these devices is very challenging. In addition, they come with two power source options: wired or battery-powered. Battery-powered ones can quickly deplete their resources while training the ML model. Moreover, development boards such as Raspberry Pi are generally referred to as IoT prototyping devices [16], and they can run ML models as they are small computer systems.

4.2. Inference approaches

The inference part of an on-device ML algorithm can be in two ways: Conventional or TinyML. Below, we explain the details of these two methods.

4.2.1. Conventional

Conventionally, the pre-trained model is shared with the local devices in plain formats. This would work for programmable IoT devices such as Raspberry Pi as they are small size computers and allow to run the ML algorithm. However, this approach cannot be extended to commercial-of-the-shelf IoT devices, smartphones, or boards like Arduino.

4.2.2. TinyML

Tiny Machine Learning (TinyML) is an approach that brings ML inference to tiny IoT devices such as Micro Controller Units (MCUs). For this, TinyML converts ML models into C code. To this end, many TinyML tools and libraries have been developed in the industry for model conversion such as TensorFlow Lite by Google [17], Embedded Learning Library (ELL) by Microsoft [18], ARM-NN [19], and Common Microcontroller Software Interface Standard NN (CMSIS-NN) by ARM [20]. Additionally, various open-source TinyML frameworks such as EmbML [21], emlearn [22], and MicroMLGen [23] are proposed in the literature. In this work, we use MicroMLGen library [23] since it can port a wide range of supervised learning algorithms such as LR, RF, DT and NB.

5. Experiment methodology

5.1. Dataset

In this work, we use a dataset of IoT network traffic captured in a Distributed Smart Space Orchestration System (DS2OS) environment [24]. The dataset covers 4 different sites: a house with 6 rooms, a 2-rooms flat, a 3-rooms flat, and an office with 10 rooms. Each site includes 7 different service types that manage the different IoT devices such as light controllers, door locks, thermostats, washing machines, batteries, motion sensors, and smartphone. The dataset includes 357 953 samples in total.

5.1.1. Types of attacks in the dataset

The DS2OS dataset was created to detect the anomalies that are caused by seven different types of attacks: DoS, network scan, malicious control, malicious operation, spying, data probing, and wrong setup. A DoS attack makes all the resources exhausted with the continuous request while a network scan anomaly occurs when someone pretends to a client to access the network. Additionally, a malicious control anomaly occurs when someone out of the network controls the network traffic and malicious operation transpires when the attackers alter the original activities. Moreover, the spying anomaly takes place when someone tracks the activities leaking the crucial information and the data type probing occurs when the transmitted data is changed. Finally, the wrong setup anomaly occurs when someone attempt to change the setup or the user does not configure correctly.

5.1.2. Preprocessing the dataset

The DS2OS dataset consists of categorical data that have numerical and nominal values. In total, there exist 13 features: sourceID, sourceAddress, sourceType, sourceLocation, destinationServiceAddress, destinationServiceType, destinationLocation, accessedNodeAddress, accessedNodeType, operation, value, timestamp and normality. We removed the timestamp column since it has minimal correlation with dependent variable “normality”. There were some missing data in “accessedNodeType” feature and we filled them with the mode of this feature. Similarly, we filled the missing data in the “value” feature with 0 and unexpected string data such as “false”, “true”, “none” and “twenty” with 0, 1, 0 and 20, respectively. After feature selection and filling the missing data, we converted categorical data into numeric form by using the label encoding.

Table 1
Parameters used for each ML algorithm.

ML Algorithm	Parameters
LR	solver='liblinear'
k-NN	n_jobs=1, n_neighbors = 5, metric='Minkowski', p = 2
DT	criterion='gini', max_depth=None
RF	n_jobs=1, criterion='gini', max_depth=None
NB	var_smoothing=1e−9
ANN	activation='relu', solver='adam' , hidden_layer_sizes = (100,)

Table 2
Dataset accuracy results.

ML Algorithm	Accuracy	Precision	Recall	F1-score
LR	0.98	0.98	0.90	0.94
k-NN	0.99	0.99	0.96	0.98
DT	0.99	0.99	0.96	0.98
RF	0.99	0.99	0.96	0.98
NB	0.97	0.99	0.81	0.89
ANN	0.99	0.98	0.94	0.97

Table 3
Device specifications used in the experiments.

Approach	Device	Hardware	Software
Cloud	Azure Cloud	Intel Xeon 16 core CPU 32 GB RAM	Scikit-learn
Edge	Dell Laptop	Intel(R) Core(TM) i7-9750H CPU 16 GB RAM	Scikit-learn
IoT	Raspberry Pi 4	Quad core Cortex-A72 64-bit SoC 8 GB RAM	Scikit-learn
IoT end device	Azure IoT Kit	STMicroelectronics STM32F412 256 KB RAM	MicroMLGen

5.1.3. Dataset accuracy

As noted, we implemented all ML algorithms using the popular Python scikit-learn library [14]. The solver parameter of the LR algorithm was set to liblinear and other parameters remained as default. NB was used with the default parameters in which likelihood probability was based on Gaussian. DT algorithm was also implemented with the default parameters (i.e., criterion = gini, and max_depth = None). In k-NN, we set the n_jobs parameter to −1 and kept other parameters as default (i.e., n_neighbors = 5, Minkowski distance with p = 2). Similarly, the n_jobs parameter of the RF algorithm was set to −1 to utilize the maximum number of cores available in the device. Finally, ANN was used with default parameters (i.e., activation = relu, solver = adam, hidden_layer_sizes = (100,)). Parameters used in ML algorithms are given in Table 1.

As a baseline, we conducted several experiments to measure the accuracy of the dataset using different ML algorithms such as LR, k-NN, DT, RF, NB, and ANN. We used the metrics such as accuracy, precision, recall, and F1-score. The results of the experiment are given in Table 2. The results show that k-NN, DT, RF, and ANN outperform the LR and NB in terms of accuracy with 99%. Additionally, k-NN, DT, RF, and NB achieve 99% precision value, whereas LR and ANN achieve 98%. k-NN, DT, and RF attain 96% recall whereas ANN, LR, and NB attain 94%, 90%, and 81%, respectively. Finally, the highest F1-score is achieved by k-NN, DT, and RF, which is 98%.

5.2. Devices used

As a cloud service, we leveraged the Azure cloud computing instance that had Intel 16 cores CPU with 32 GB RAM. Additionally, we defined the edge device as a device with limited memory and computation power whereas IoT device as a device with very low memory and computation capacity. Accordingly, we used Dell computer that had Intel(R) Core(TM) i7-9750H CPU processor with 16 GB RAM as an edge, and Raspberry Pi 4 Model B with 8 GB RAM as an IoT device. Finally, we used ESP32 Azure IoT Kit as an end device to evaluate inference time thus energy consumption on this device. Table 3 shows the devices, their specifications, and the software used for the experiments for each approach in our architecture in Fig. 2.

5.3. Estimating the energy consumption of ML algorithms

We utilize two methods to estimate the energy consumption of ML algorithms: (1) Execution Time, and (2) Power Monitoring Tool.

5.3.1. Execution time

The energy consumption for processing ML algorithm can be computed using the following equation [25]:

$$E_{proc} = P_{proc} * t, \tag{3}$$

Table 4
Time consumed for training (s) of different ML model trained with different dataset size on cloud (Azure Cloud), edge (Dell Laptop) and IoT device (Raspberry Pi).

		Cloud (s)	Edge (s)	IoT (s)
Data size = 35K	LR	0.15	0.12	0.71
	k-NN	0.04	0.08	0.13
	DT	0.05	0.04	0.19
	RF	0.29	0.25	2.82
	NB	0.01	0.01	0.04
	ANN	2.46	1.86	8.09
Data size = 357K	LR	1.70	1.56	6.18
	k-NN	0.5	1.11	2.26
	DT	0.54	0.48	3.13
	RF	2.29	9.83	47.26
	NB	0.05	0.07	0.32
	ANN	331.50	171.15	419.32
Data size = 3.5M	LR	19.06	16.54	60.18
	k-NN	8.16	17.61	35.25
	DT	8.98	17.46	40.86
	RF	31.75	166.83	624.43
	NB	0.67	0.69	3.27
	ANN	1847.94	893.43	2109.90

where P_{proc} is processing power and t is the time spent for processing. In this approach, the power consumption of the device is assumed to be constant over time i.e., the power consumption of the time spent for processing (i.e., execution time) is directly proportional to the energy consumption [26]. Many studies in the literature used this approach to estimate the energy consumption of an ML-based application [27,28].

5.3.2. Power monitoring tool

Power monitoring tools enable real-time processor power estimation by providing applications, drivers, and libraries. The desktop and server providers have qualified their systems to measure the energy consumption of the hardware components with energy sensors. For instance, Intel introduces the Intel Power Gadget [29] software tool that uses Running Average Power Limit (RAPL) sensors to calculate the energy consumption of a block of code.

6. Analysis results

This section provides energy consumption analysis of on-device ML deployment approaches. The analysis was performed during the training and inference part separately. Furthermore, we compared the different ML algorithms (i.e., LR, k-NN, DT, RF, NB, and ANN) and three different data sizes. We created small, medium, and large dataset sizes by downsampling by 0.1 and oversampling by 10 of the original dataset. In the end, we obtained the dataset with 35K, 357K, and 3.5M samples to investigate the impact of dataset size on the consumed energy for various ML algorithms used for IDS. We also used two techniques to estimate the power consumption. In the following subsections, we present our analysis results using the execution time and power monitoring tool.

6.1. Energy consumption analysis via execution time

As mentioned previously in Section 5.3.1, the execution time of running the ML algorithm is directly proportional to the energy consumption. We examined the execution times of ML algorithms by taking average results of 30 runs with different seeds for three dataset sizes (i.e., 35K, 357K, 3.5M). Furthermore, we evaluated the execution times on different computational platforms (i.e., cloud, edge, and IoT device) as given in Table 3.

Table 4 shows the training time(s) of all ML algorithms using dataset sizes of 35K, 357K, and 3M on Azure Cloud, Dell Laptop, and Raspberry Pi, representing the cloud, edge, and IoT device, respectively. Our first observation is that as the dataset size increases, the training times of all ML algorithms in all computing platforms increase. Notably, the training time of ANN increases drastically with the dataset size increment. In addition, we observed that the NB training requires much less time than all other algorithms in all computing platforms, thereby consuming less energy. In addition, the RF algorithm requires significantly less training time when computed in the cloud and edge than the IoT device because the algorithm can run on multicore processors, which are supported by the cloud and edge devices. Therefore, training the RF algorithm on an IoT device is not energy-efficient. Moreover, since the computational power of Raspberry Pi is limited compared to the cloud and edge, the execution times of training are the highest for all ML algorithms.

Table 5 demonstrates the inference time (μs) for all ML algorithms on Azure Cloud, Dell Laptop, and Raspberry Pi. We observed that the inference time of the k-NN algorithm increases significantly as the dataset size increases. The reason for that, the algorithms calculate the distance between the new data instance and each data sample in the dataset. In addition, the inference while using the k-NN algorithm takes a very long time on all computing platforms. Thus, it is not an energy-friendly solution for battery-powered

Table 5
Time consumed for inference time (μs) of different ml model trained with different dataset size on cloud (Azure Cloud), edge (Dell Laptop) and IoT device (Raspberry Pi).

		Cloud (μs)	Edge (μs)	IoT (μs).
Data size = 35K	LR	0.10	0.20	0.69
	k-NN	34.10	27.18	121.97
	DT	0.11	0.10	0.51
	RF	4.00	2.86	18.19
	NB	0.14	0.14	2.44
	ANN	0.50	1.00	6.54
Data size = 357K	LR	0.05	0.07	0.35
	k-NN	81.40	57.78	298.50
	DT	0.07	0.09	0.33
	RF	0.80	1.53	10.11
	NB	0.10	0.30	2.07
	ANN	1.35	3.07	5.37
Data size = 3.5M	LR	0.04	0.04	0.04
	k-NN	513.31	530.14	2078.01
	DT	0.07	0.09	0.34
	RF	0.62	1.88	10.10
	NB	0.19	0.34	2.13
	ANN	1.28	2.97	5.58

Table 6
Execution times (inference time (μs)) of different ML algorithms on IoT end device.

Azure IoT KiT			
Inference (μs)			
LR	DT	RF	NB
73.06	65.06	2088	602

IoT devices to enable on-device ML intrusion detection applications. Furthermore, we converted the pre-trained ML model at the edge device by utilizing the TinyML approach (i.e., using the microMLGen library). Later, we deployed the converted ML model to the IoT device (i.e., Azure IoT Kit) to perform inference on-device. The inference time is given in Table 6. DT obtains the best result, which performs the inference in 65.06 μs. Again, it is because the DT algorithm does not perform multiplication operations.

6.2. Energy consumption analysis via power monitoring tool

In this subsection, we provide an evaluation of energy consumption in Joules and power estimation in Watt while training the ML algorithms at the edge (i.e., Dell Laptop). We leverage the Intel Power Gadget software tool [29] for evaluations. Table 7 depicts the energy consumption (J) and average power consumption (W) of different ML algorithms trained with different dataset sizes. We observed a similar trend in results as obtained with execution times in the previous subsection. NB achieves the least energy consumption in training for all dataset sizes (i.e., 2.44 J, 3.73 J, 15.16 J for 35K, 357K, and 3.5M, respectively). It is followed by DT, k-NN, and LR, respectively. RF consumes relatively higher energy than DT since the algorithm processes several decision trees to train the model. Additionally, it is clearly seen that ANN consumes the highest energy in training. Therefore, ANN is not an energy-efficient algorithm for training it on-device, and it also may cause significant energy consumption for cloud platforms in the case of huge datasets. Table 7 also shows the total energy consumption to infer the test dataset. k-NN consumes the highest energy during the inference since it is a lazy learner and computationally intensive in the inference part. ANN consumes considerably more power than other algorithms during training since it requires higher computation that includes many multiplication and addition operations. On the contrary, the average power usage of DT and RF (i.e., tree-based algorithms) is the lowest because they perform computation without any multiplication operations. On the other hand, power estimation is inconsistent when computation takes a short time, such as less than a few seconds. That is why we observe relatively higher power consumption during inference.

6.3. Computational complexity analysis

In this subsection, we analyze the impact of the computational complexity of the algorithms on power consumption. Table 8 illustrates the training and inference computational complexity of ML algorithms based on sckit-learn implementations [7,14]. Our overall results show that the higher computational complexity of the ML algorithms results in more power consumption. Moreover, the computational complexity is proportional with respect to the number of training samples and input features. NB is better in terms of training complexity compared to others. The training complexity of LR regression increases exponentially as the number of input features increases. In addition, DT has the least computational complexity in inference. Besides the number of training samples and input features, the number of coefficients and hidden layers affect the computational complexity in ANN. A network having a large number of coefficients and hidden layers require more computation to train a model. Therefore, as the number of

Table 7
Energy Consumption (J) and Average Power Consumption (W) of different ml model trained with different dataset size on edge (Dell Laptop).

		Training		Inference	
		Energy (J)	Power (W)	Energy (J)	Power (W)
Data size = 35K	LR	4.53	32	2.36	73
	k-NN	3.52	39	11.40	36
	DT	3.1	42	1.68	51
	RF	20.68	63	4.78	51
	NB	2.44	48	2.19	50
	ANN	103.17	52	2.81	64
Data size = 357K	LR	37.86	26	1.25	24
	k-NN	29.21	28	338.94	53
	DT	14.97	28	1.99	52
	RF	173.68	16	16.16	71
	NB	3.73	41	2.85	48
	ANN	9786.03	37	20.37	60
Data size = 3.5M	LR	361.07	23	4.73	53
	k-NN	322.72	19	14 200	24
	DT	178.73	11	4.29	35
	RF	1967.51	11	130.81	63
	NB	15.16	25	8.41	27
	ANN	22 937.68	36	157.14	55

Table 8
Computational complexity of ML algorithms.

ML Algorithm	Training	Inference
LR	$O(mn^2 + m^3)$	$O(n)$
k-NN	–	$O(mn)$
DT	$O(mnlog(m))$	$O(log(m))$
RF	$O(N_t mnlog(m))$	$O(N_t log(m))$
NB	$O(mn + nc)$	$O(nc)$
ANN	$O(mnh^k i)$	$O(n)$

m:number of training samples, n: number of input features, N_t: number of tree, c: number of classes, h: number of coefficients, k: number of hidden layers, i: number of iterations.

coefficients and hidden layers increases, the power consumption increases because of the higher computational complexity. Since training time is a critical factor for on-device ML, a single hidden layer should be used if the accuracy of the model is satisfied.

6.4. Comparative analysis

Additionally, we comparatively discuss the results of on-device ML in terms of accuracy, energy consumption, and latency. Fig. 3 demonstrates the comparative analysis for the edge device with the 3.5M dataset size. Among all ML algorithms, k-NN, DT, and RF give the best performance results in terms of accuracy, precision, recall, and F1-Score. However, k-NN consumes higher energy during inference which is not preferable for IoT devices. DT consumes less energy during the training and inference part than RF due to the increasing number of trees in the RF algorithm. Therefore, the DT algorithm provides better accuracy performance and energy efficiency. On the other hand, the response time of the ML models are critical for real-time IDS. In conventional cloud computing-based ML models, the communication and computation latency affect the application response time. Therefore, performing ML models on-device becomes imperative.

7. Related work

In the last decade, ML-based IDS for IoT applications have attracted significant attention due to their high accuracy and efficiency. Recent surveys [30–34] provide an overview of ML-based intrusion detection solutions for IoT applications. In [35–37], the authors examine the performance of ML algorithms on IDS for IoT applications and discuss how to choose ML algorithms according to specific application needs.

Cloud Computing-based ML. Some of the studies in the literature particularly focused on cloud-based ML models for IDS. Morfino and Rampone [38] investigate the performance of supervised ML algorithms in terms of accuracy and training times by using Apache Spark in the cloud. They propose an approach that performs cloud training and deploys the model on IoT devices. Karende and Joshi [39] develop an attack detection system by using BigQuery ML (BQML) model from Google cloud. Anthi et al. [40]

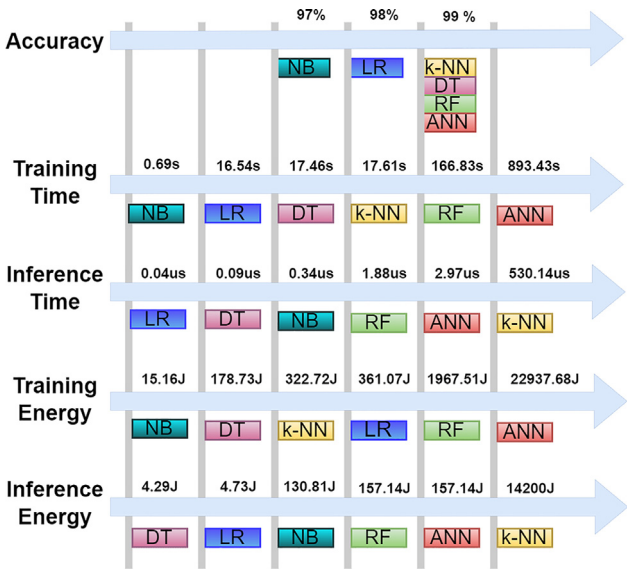


Fig. 3. Representation of comparative analysis for edge device.

present IDS to detect wireless network attacks such as DoS, replay, man-in-the-middle, and reconnaissance and classify the attack type. Further, they classify the type of IoT devices located in the network.

Edge Computing-based ML. As the IoT devices, their applications, and their generated data exponentially increase, high computation demand and network congestion occur in the cloud. As a result, high latency becomes inevitable for cloud computing systems. Edge computing has emerged to meet low latency requirements and reduce network congestion [41]. A recent survey [42] demonstrates the edge-based IoT architecture and reviews research efforts on edge-based solutions for IoT security such as IDS and privacy-preserving systems. Kumar et al. [43] develop edge ML architecture for detecting intrusions in IP-based IoT deployments to minimize latency. They evaluate XGBoost, Support Vector Machine (SVM), and Logistic Regression (LR) algorithms on edge devices (i.e., Raspberry Pi3) by using Azure edge porting. Jian et al. [44] develop an intrusion prevention model based on edge computing to detect malicious activities in real-time for smart homes. They evaluate the decision tree J48 algorithm on Raspberry Pi as an edge device. Eskandari et al. [45] design a platform-independent anomaly-based IDS to execute it on resource-constraint edge devices. They implement two one-class classification algorithms such as isolation forest and local outlier factor to detect HTTP and SSH brute force, port scanning, and SYN flood attack. Basically, the isolation forest separates the anomaly instances by exploiting different characteristics of these instances. The local outlier factor is based on density that calculates density by measuring distances between data instances.

Hybrid ML Approaches. Hybrid systems (i.e., edge–cloud) have been proposed to take advantage of both cloud and edge computing. Alghamdi and Bellaiche [46] present a combined ML model for IDS based on both edge and cloud to address the limitation of the cloud such as network bandwidth and delay. Their proposed model reduces the training time while improving the accuracy. Similarly, Cassales et al. [47] leverage both edge and cloud computing advantages. Since the classification is performed at the edge, the latency is reduced. On the contrary, the ML model is improved and updated on the cloud. In addition, Roy et al. [48] develop fog–cloud infrastructure for intrusion detection systems to lower delay and provide energy efficiency. Wang et al. [49] analyze computational performance of Deep Neural Network (DNN) models for image recognition on both cloud computing platforms (i.e., Google Colab) and edge computing platforms (i.e., Intel NCS2).

Tiny ML. TinyML is an innovative approach that enables running inference on IoT end devices (i.e., Micro-controller Units (MCUs)). Ren et al. [50] propose TinyML with online learning (TinyOL) that empowers the incremental training of the pre-trained ML model on resource-constraint IoT devices (e.g., Arduino Nano 33 BLE board). They exploit the TinyOL to execute a pre-trained neural network model for anomaly detection in industrial IoT.

Our differences. To the best of our knowledge, no study in the literature provides a comparative analysis of energy consumption of on-device ML deployment approaches for IDS in SHSs. In this work, we investigate the energy efficiency of the most common ML algorithms such as LR, k-NN, DT, RF, NB, and ANN used in IDS on SHSs. In addition, we compare the ML-based IDS performed on cloud, edge, and IoT end devices in terms of energy consumption.

8. Conclusions

In conclusion, on-device ML models for IDS have gained tremendous popularity due to their promising solutions to real-time response and privacy-preserving features. However, performing energy-intensive ML tasks on-device is a critical constraint that needs to be

investigated. In this work, we evaluated the different deployment approaches to ML-based intrusion detection applications in terms of energy consumption. Particularly, we compared various on-device ML deployment approaches such as cloud computing-based, edge computing-based, and IoT device-based for training. Additionally, we examined the conventional and tinyML approaches used to perform inference on IoT end devices.

Our main conclusions are listed as follows:

- NB algorithm requires less training time than all of the other ML algorithms for all platforms and dataset sizes. However, we also found that the accuracy of NB (i.e., 97%) is also less than other algorithms. Therefore, one should consider a trade-off between accuracy and execution time while deploying a real-life system.
- The energy consumption of k-NN during inference rapidly increases as the dataset size increases (i.e., 11.40 J, 338.94 J, and 14 200 J for 35K, 357K, and 3.5M, respectively). Therefore, it may not be ideal especially for the IoT devices.
- The power estimation results showed that DT and RF consume less power (i.e., 11 W) during training with the 3.5M dataset size as well as achieve better accuracy (i.e., 99%). However, the execution time of RF for training and inference is higher than DT. Therefore, using RF may not be preferable due to higher energy consumption unless it does not provide better accuracy.
- The cloud computing-based ML that offers multiple cores is efficient in terms of execution time for the algorithms which can run on multicore in parallel such as RF, but it does not decrease the execution time of ANN so much, which is not supporting multicore processing.
- DT gives the best performance in terms of inference time (i.e., 65.06 μs) when performed in Azure IoT KiT.

Besides the computational energy, communication energy should be considered in both cloud computing-based ML and edge computing-based ML approaches. In both approaches, data needs to be transmitted by IoT devices to a central server (i.e., cloud or edge devices) to train the model. Especially, in real-time applications such as intrusion detection, continuous transmission of large volumes of data causes energy consumption. On the other hand, communication cost arises when IoT devices on the network need to exchange parameters/data while running ML distributedly on-device. Therefore, the transmission power consumption of IoT devices is an important factor to consider in the design and implementation of machine learning systems, and further research into this area could help to improve the energy efficiency of these systems. Our results can be used as a guide while deploying a real-world IoT intrusion detection system and further point to a research direction (e.g., the energy consumption of on-device ML models) that needs improvement.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the U.S. National Science Foundation (Award: NSF-CAREER CNS-1453647) and Microsoft Research Grant. Dr. N. Tekin was supported by Scientific and Technological Research Council of Turkey (TUBITAK-BIDEB) 2219—International Postdoctoral Research Scholarship Program. The views expressed are those of the authors only, not of the funding agencies.

References

[1] H. Lin, N.W. Bergmann, IoT privacy and security challenges for smart home environments, Information 7 (3) (2016) 44.
[2] V. Tamilselvi, S. Sribalaji, P. Vigneshwaran, P. Vinu, J. GeethaRamani, IoT based health monitoring system, in: 6th International Conference on Advanced Computing and Communication Systems, ICACCS, IEEE, 2020, pp. 386–389.
[3] A.K. Sikder, G. Petracca, H. Aksu, T. Jaeger, A.S. Uluagac, A survey on sensor-based threats and attacks to smart devices and applications, Commun. Surv. Tutorials 23 (2) (2021) 1125–1159.
[4] A.K. Sikder, L. Babun, H. Aksu, A.S. Uluagac, Aegis: A context-aware security framework for smart home systems, in: Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC, ACM, 2019, pp. 28–41.
[5] Z.B. Celik, L. Babun, A.K. Sikder, H. Aksu, G. Tan, P. McDaniel, A.S. Uluagac, Sensitive information tracking in commodity {IoT}, in: 27th USENIX Security Symposium, ACM, 2018, pp. 1687–1704.
[6] K. Cao, Y. Liu, G. Meng, Q. Sun, An overview on edge computing research, Access 8 (2020) 85714–85728.
[7] S. Dhar, J. Guo, J. Liu, S. Tripathi, U. Kurup, M. Shah, A survey of on-device machine learning: An algorithms and learning theory perspective, Trans. Internet Things 2 (3) (2021) 1–49.
[8] L. Dutta, S. Bharali, TinyML meets IoT: A comprehensive survey, Internet of Things 16 (2021) 100461.
[9] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J.A. Halderman, L. Invernizzi, M. Kallitsis, et al., Understanding the mirai botnet, in: 26th USENIX Security Symposium, 2017, pp. 1093–1110.
[10] H. Oz, A. Aris, A. Levi, A.S. Uluagac, A survey on ransomware: Evolution, taxonomy, and defense solutions, ACM Comput. Surv. (2021).
[11] S. Bhansali, A. Aris, A. Acar, H. Oz, A.S. Uluagac, A first look at code obfuscation for WebAssembly, in: Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks, 2022, pp. 140–145.
[12] A. Acar, H. Fereidooni, T. Abera, A.K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, S. Uluagac, Peek-a-boo: I see your smart home activities, even encrypted!, in: Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 207–218, http://dx.doi.org/10.1145/3395351.3399421.
[13] T. Dietrich, Smart home product security risks can be alarming, 2019, https://www.insurancejournal.com/news/national/2019/01/03/513394.htm, [Online; accessed 01-June-2022].

[14] 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, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[15] A.H. Jahromi, M. Taheri, A non-parametric mixture of Gaussian naive Bayes classifiers based on local independent features, in: Artificial Intelligence and Signal Processing Conference, AISP, IEEE, 2017, pp. 209–212.

[16] C. Perera, C.H. Liu, S. Jayawardena, M. Chen, A survey on internet of things from industrial market perspective, Access 2 (2014) 1660–1679.

[17] TensorFlow, tflite-micro, 2022, https://github.com/tensorflow/tflite-micro, [Online; accessed 2-February-2022].

[18] Microsoft, ELL, 2020, https://github.com/microsoft/ELL, [Online; accessed 2-February-2022].

[19] MikeJKelly, ARM-software, 2022, https://github.com/ARM-software/armnn, [Online; accessed 2-February-2022].

[20] GuentherMartin, ARM-software, 2021, https://github.com/ARM-software/CMSIS_5, [Online; accessed 2-February-2022].

[21] R. Sanchez-Iborra, A.F. Skarmeta, Tinyml-enabled frugal smart objects: Challenges and opportunities, Circuits Syst. Mag. 20 (3) (2020) 4–18.

[22] Jonnor, emlearn, 2021, https://github.com/emlearn/emlearn, [Online; accessed 2-February-2022].

[23] eloquentarduino, MicroMLGen, 2020, https://github.com/eloquentarduino/micromlgen, [Online; accessed 2-February-2022].

[24] F. Aubet, M. Pahl, DS2OS traffic traces, 2018, https://www.kaggle.com/datasets/francoisxa/ds2ostraffictraces, [Online; accessed 22-April-2022].

[25] N. Tekin, V.C. Gungor, Analysis of compressive sensing and energy harvesting for wireless multimedia sensor networks, Ad Hoc Netw. 103 (2020) 102164.

[26] N. Tekin, H.U. Yildiz, V.C. Gungor, Node-level error control strategies for prolonging the lifetime of wireless sensor networks, IEEE Sens. J. 21 (13) (2021) 15386–15397.

[27] I. Amezzane, A. Berrazzouk, Y. Fakhri, M. El Aroussi, M. Bakhouya, Energy consumption of batch and online data stream learning models for smartphone-based human activity recognition, in: 4th World Conference on Complex Systems, WCCS, IEEE, 2019, pp. 1–5.

[28] M. Ferro, G.D. Silva, F.B. de Paula, V. Vieira, B. Schulze, Towards a sustainable artificial intelligence: A case study of energy efficiency in decision tree algorithms, Concurr. Comput.: Pract. Exper. (2021) e6815.

[29] P.C.K. Timothy McKay, Intel power gadget, 2019, https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html, [Online; accessed 28-June-2022].

[30] A.S. Dina, D. Manivannan, Intrusion detection based on machine learning techniques in computer networks, Internet of Things 16 (2021) 100462.

[31] S. Abdelhamid, M. Aref, I. Hegazy, M. Roushdy, A survey on learning-based intrusion detection systems for IoT networks, in: Tenth International Conference on Intelligent Computing and Information Systems, ICICIS, IEEE, 2021, pp. 278–288.

[32] J. Long, F. Fang, H. Luo, A survey of machine learning-based IoT intrusion detection techniques, in: 6th International Conference on Smart Cloud (SmartCloud), IEEE, 2021, pp. 7–12.

[33] E.P. Nugroho, T. Djatna, I.S. Sitanggang, A. Buono, I. Hermadi, A review of intrusion detection system in IoT with machine learning approach: current and future research, in: 6th International Conference on Science in Information Technology (ICSITech), IEEE, 2020, pp. 138–143.

[34] R. Ahmad, I. Alsmadi, Machine learning approaches to IoT security: A systematic literature review, Internet of Things 14 (2021) 100365.

[35] M. Almseidin, M. Alzubi, S. Kovacs, M. Alkasassbeh, Evaluation of machine learning algorithms for intrusion detection system, in: 15th International Symposium on Intelligent Systems and Informatics, SISY, IEEE, 2017, pp. 000277–000282.

[36] A. Verma, V. Ranga, Machine learning based intrusion detection systems for IoT applications, Wirel. Pers. Commun. 111 (4) (2020) 2287–2310.

[37] S.S.S. Sugi, S.R. Ratna, Investigation of machine learning techniques in intrusion detection system for IoT network, in: 3rd International Conference on Intelligent Sustainable Systems, ICISS, IEEE, 2020, pp. 1164–1167.

[38] V. Morfino, S. Rampone, Towards near-real-time intrusion detection for IoT devices using supervised learning and apache spark, Electronics 9 (3) (2020) 444.

[39] J. Karande, S. Joshi, Real-time detection of cyber attacks on the IoT devices, in: 11th International Conference on Computing, Communication and Networking Technologies, ICCCNT, IEEE, 2020, pp. 1–6.

[40] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, P. Burnap, A supervised intrusion detection system for smart home IoT devices, Internet Things J. 6 (5) (2019) 9042–9053.

[41] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, Internet Things J. 3 (5) (2016) 637–646.

[42] K. Sha, T.A. Yang, W. Wei, S. Davari, A survey of edge computing-based designs for IoT security, Digit. Commun. Netw. 6 (2) (2020) 195–202.

[43] H. Kumar, A.R. Jadhav, G. Sasirekha, J. Bapat, D. Das, Intelligent edge detection of attacks on IP-based IoT deployments, in: 19th OITS International Conference on Information Technology, OCIT, IEEE, 2021, pp. 132–137.

[44] C. Jiang, J. Kuang, S. Wang, Home IoT intrusion prevention strategy based on edge computing, in: 2nd International Conference on Electronics and Communication Engineering, ICECE, IEEE, 2019, pp. 94–98.

[45] M. Eskandari, Z.H. Janjua, M. Vecchio, F. Antonelli, Passban IDS: An intelligent anomaly-based intrusion detection system for IoT edge devices, Internet Things J. 7 (8) (2020) 6882–6897.

[46] R. Alghamdi, M. Bellaiche, A deep intrusion detection system in lambda architecture based on edge cloud computing for IoT, in: 4th International Conference on Artificial Intelligence and Big Data, ICAIBD, IEEE, 2021, pp. 561–566.

[47] G.W. Cassales, H. Senger, E.R. de Faria, A. Bifet, IDSA-IoT: an intrusion detection system architecture for IoT networks, in: Symposium on Computers and Communications, ISCC, IEEE, 2019, pp. 1–7.

[48] S. Roy, J. Li, Y. Bai, A two-layer fog-cloud intrusion detection model for IoT networks, Internet Things (2022) 100557.

[49] X. Wang, F. Zhao, P. Lin, Y. Chen, Evaluating computing performance of deep neural network models with different backbones on IoT-based edge and cloud platforms, Internet Things 20 (2022) 100609.

[50] H. Ren, D. Anicic, T.A. Runkler, The synergy of complex event processing and tiny machine learning in industrial IoT, in: 15th International Conference on Distributed and Event-Based Systems, DEBS, ACM, 2021, pp. 126–135.



Nazli Tekin received the B.S. degree in computer engineering from Koc University, Istanbul, Turkey, in 2011, and the Ph.D. degree in electrical and computer engineering from Abdullah Gül University, Kayseri, Turkey, in 2020. She is an Assistant Professor with the Department of Software Engineering, Erciyes University, Kayseri. Currently, she is post-doctoral associate at the Cyber-Physical Systems Security Lab, Department of Electrical and Computer Engineering, Florida International University, Miami, FL, USA. Her research interests include wireless sensor networks, IoT security.



Abbas Acar received his MSc and Ph.D. degrees in the Department of Electrical and Computer Engineering at Florida International University in 2019 and 2020, respectively. Before that, he received his B.S. degree in Electrical and Electronics Engineering from Middle East Technical University in 2015. His research interests include continuous authentication, IoT security/privacy, and homomorphic encryption. More information can be obtained from https://web.eng.fiu.edu/aacar/



Ahmet Aris is a Research Assistant Professor in the Department of Electrical and Computer Engineering at Florida International University. He is conducting research in Cyber-Physical Systems Security Lab (CSL) at Florida International University under the supervision of Dr. A. Selcuk Uluagac. He earned both PhD and MSc. in Computer Engineering from the Graduate School of Science, Engineering and Technology at Istanbul Technical University, Turkey. He also worked at Medianova CDN R&D Center as an R&D Analyst. In addition, he conducted research in the Networked Embedded Systems (NES) Group at Swedish Institute of Computer Science (SICS) as a visiting researcher. His research interests include IoT Security, Network Security, Web Security, and Malware.



A. Selcuk Uluagac is currently an Eminent Scholar Chaired Associate Professor in the Department of Electrical and Computer Engineering at FIU, where he leads the Cyber-Physical Systems Security Lab, with an additional courtesy appointment in the Knight Foundation School of Computing and Information Science. Before FIU, he was a Senior Research Engineer at Georgia Tech and Symantec. He holds a PhD from Georgia Tech and MS from Carnegie Mellon University. He received US National Science Foundation (NSF) CAREER Award (2015), Air Force Office of Sponsored Research's Summer Faculty Fellowship (2015), and University of Padova (Italy)'s Faculty Fellowship (2016), and Google's ASPIRE Research award in security and privacy (2021). He is an expert in the areas of cybersecurity and privacy with an emphasis on their practical and applied aspects and teaches classes in these areas. He has hundreds of research papers/studies/publications in the most reputable venues. His research in cybersecurity and privacy has been funded by numerous government agencies and industry, including the US NSF, the US Dept. of Energy, US Air Force Research Lab, US Dept. of Labor, Cyber Florida, Google, Microsoft, Trend Micro, and Cisco, inter alia. He is very entrepreneurial and visionary with his research. Many of his research ideas have resulted in patents with one licensed to a company recently. He has served on the program committees of top-tier security conferences such as IEEE Security & Privacy ("Oakland"), NDSS, Usenix Security, inter alia. He was the General Chair of ACM Conference on Security and Privacy in Wireless and Mobile Networks (ACM WiSec) in 2019. Currently, he serves on the editorial boards of IEEE Transactions on Mobile Computing, Elsevier Computer Networks Journal, and the IEEE Communications and Surveys and Tutorials (network security lead).



V. Cagri Gungor received his Ph.D. degree in electrical and computer engineering from the Broadband and Wireless Networking Laboratory, Georgia Institute of Technology, Atlanta, GA, USA, in 2007. Currently, he is a Full Professor and Dean of Faculty of Computer Science, Abdullah Gul University in Kayseri, Turkey. His current research interests are in machine learning, machine-to-machine communications, next-generation wireless networks, wireless ad hoc and sensor networks. Dr. Gungor has authored more than 100 papers in refereed journals and international conference proceedings, and has been serving as an editor, reviewer and program committee member to numerous journals and conferences in these areas. He is also the recipient of TUBITAK Young Scientist Award in 2017, Science Academy Young Scientist Award (BAGEP) in 2017, Turkish Academy of Sciences Distinguished Young Scientist Award (TUBA-GEBIP) in 2014, the IEEE Trans. on Industrial Informatics Best Paper Award in 2012, the European Union FP7 Marie Curie RG Award in 2009.