

TinyIDS - An IoT Intrusion Detection System by Tiny Machine Learning

Pietro Fusco(✉), Gennaro Pio Rimoli, and Massimo Ficco

Università degli Studi di Salerno, Via Giovanni Paolo II, 132 - 84084 Fisciano, SA, Italy
{pfusco,grimoli,mficco}@unisa.it

Abstract. The use of Internet of Things (IoT) devices in sectors, such as healthcare, automotive, and industrial automation, has increased the risk of attacks against critical assets. Machine learning techniques may be utilized to identify malicious behaviors, but they often require dedicated, energy-intensive, and expensive devices, which may not be deployable in IoT infrastructures. Furthermore, privacy constraints, security policies, and latency constraints could limit the sending of sensitive data to powerful remote servers. To address this issue, the emerging field of TinyML offers a solution for implementing machine learning algorithms directly on resource-constrained devices. Therefore, this article presents the implementation of an intrusion detector, named TinyIDS, which exploits the Tiny machine learning techniques. The detector can be deployed on resource-constrained IoT devices to detect attacks against sensor networks, as well as malicious behaviors of compromised smart objects. On-board training has been exploited to train and analyze data locally without having to transfer sensitive data to remote or untrusted cloud services. The solution has been tested on common MCU-based devices and ToN_IoT datasets.

Keywords: TinyML · On-Board Training · IDS · Security · Intrusion Detection · MCU · Microcontroller

1 Introduction

The widespread diffusion of consumer electronics in every context of our daily lives, including healthcare, transport, industrial automation, military, etc., has ushered in a new era, which transcends individual functionality, making it an integrated element of a more complex interconnected Internet of Things (IoT) ecosystem [1]. Despite the great potential, their utilization in the context in which sensitive data are managed makes them prone to cyber intrusion due to the lack of robust security measures implementation. Several studies described potential attacks on the different layers of an IoT system, highlighting the risks that a vulnerability in IoT devices could pose [2–4]. Such vulnerabilities may result in the theft of sensitive data, as well as be exploited to cause system malfunctions.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024 Workshops, LNCS 14816, pp. 71–82, 2024.
https://doi.org/10.1007/978-3-031-65223-3_5

Centralized Machine Learning (ML) based security detection mechanisms may not be suitable solutions for such contexts, in which strict security and privacy policies are required, as they could demand sensitive data to be transferred and processed in remote and untruest servers. For example, in healthcare applications, smart monitoring sensors worn by the patients should be able to read and process private data (such as heartbeat, blood pressure, and sugar level) and notify alerts directly to the interested party (*e.g.*, if the values are symptomatic of some ongoing pathology), without sending data to a cloud service. Moreover, there are application contexts in which may be necessary to deploy smart devices in areas with poor or no Internet access (*e.g.*, along drinking water pipes), as well as provide low-latency and low-energy consumption (*e.g.*, Drone of Things).

Tiny machine learning (TinyML) emerges as a pivotal technology to process ML algorithms on low-cost devices [5]. Deploying TinyML algorithms on devices that are the source of data or nearby can significantly improve privacy, security data protection regulations, and achieve low latency with comparable accuracy.

This paper presents an ML-based implementation of a Tiny Intrusion Detection System (TinyIDS) to be installed on commercial resource-constrained Microcontroller Units (MCUs) based devices. TinyIDS can be used to detect security anomalies within a network of IoT objects, with one of them assuming the role of intrusion detector. Specifically, two kinds of intrusion detector models could be implemented: rule-based and machine learning-based. Rule-based IDS uses parameters such as signatures, message content, protocols, logs, etc. to detect threats. On the other hand, a machine learning-based model detects anomalous behaviors and dynamically adapts to new threats, making them more efficient compared to traditional methods. In this paper, we focused on the latter approach. Moreover, on-board training has been adopted to avoid transferring sensitive data to remote cloud services and improve security and privacy. An extension of Genann C library implemented in our previous work has been used [6].

TinyIDS can be used to detect attacks against a sensor network or malicious behaviors of smart objects (*e.g.*, the anomalous behavior of a compromised drone within a swarm). The device chosen as a detector can be either a peer node in the network (*e.g.*, the sink node), or the most powerful device in terms of energy, memory, and computational resources (*e.g.*, the leader drone within the swarm).

An experimental campaign has been carried out, deploying the implemented detector on common MCU-based devices, including ESP32, ESP8266, and Arduino Portenta H7. ToN_IoT datasets have been used for performance evaluation. It includes heterogeneous IoT data sources collected from telemetry and industrial sensors [7], along with several attack traces. Experimental results have shown that the proposed solution archives about 85% of accuracy.

The paper is structured as follows: Sect. 2 describes the used dataset and the current state of the art. Dataset pre-processing is presented in Sect. 3. Section 4 presents the implementation of the proposed solution. The solution evaluation is shown in Sect. 5. Finally, potential future work based on the results obtained is outlined in Sect. 6.

2 Background and Related Work

2.1 TON_IoT Dataset

The TON_IoT dataset was generated in 2019, and it was based on a testbed environment at the Cyber Range and IoT Labs at the University of New South Wales (UNSW) Canberra, Australia[1] The experimental setup specified in [7] includes physical and simulated connected devices. The name of the dataset "TON_IoT" refers to its content and provenance: telemetry data, operating system data, and network data from the IoT/IIoT testbed. Three layers of virtualized switches and routers were integrated into the testbed:

- Cloud layer (virtualized): Contains a Message Queuing Telemetry Transport (MQTT) broker.
- Fog layer (virtualized): Contains the monitored vulnerable hosts and attacker machines. It includes a Node-RED [8] server that simulates 6 out of 7 sensor sources.
- Edge layer (physical): Contains the physical devices, from which an ESP8266 weather station acts as the 7th sensor data source. Other devices in the edge layer are two smartphones, a smart TV, and the NSX VMware server emulating virtualized layers.

As an objective to acquire the specifications of IoT and IIoT systems, this paper aims to focus on the IoT telemetry part of this dataset. TON_IoT telemetry dataset makes available the sensor readings as registered by a Node-RED server, delivered through the MQTT protocol via the MQTT broker in the Cloud layer. This part of the data is provided for any sensors as a comma-separated values (CSV) file. Network capture and processing data from operating systems on the targeted virtual machines, which will not be used in this paper, are also part of the dataset.

There is a high degree of heterogeneity in the dataset. To gain a better awareness of its content, a closer look at its characteristics can help. Specifically, the data falls into these classes:

- IoT telemetry data (*i.e.*, IoT/IIoT sensor measurements, which will be used for the scope of this paper);
- Linux host data;
- Windows host data.

Additionally, basic specifications for security events are available. The dataset also makes available basic statistics on the number of records in each class. The TON_IoT dataset comprises a hierarchical structure where are included the following directories:

- *Raw datasets*: IoT/IIoT datasets were logged in log and CSV files, where more than 10 IoT and IIoT sensors, such as weather and Modbus sensors, were used to capture their telemetry data;

[1] UNSW - Canberra: https://www.unsw.adfa.edu.au.

- *Processed datasets*: To produce standard features and labels, the four datasets that were previously included in these directories have been filtered (to be used on any platform, the whole dataset has been processed and filtered in CSV format);
- *Train_Test_datasets*: Samples of the four datasets in CSV format that were selected for evaluating the fidelity and efficiency of new cyber security application-based AI and machine learning algorithms are included in this folder.

As previously described, the telemetry data come from 7 physical and simulated sensors. Table 1 shows a summary of the sensors and their outputs. There are 17 features from seven different sensors in both the Processed Dataset and Train_TEST_Dataset dataset, as well as a time stamp. A binary label field has been added to each data record, (0 means normal operation, and 1 indicates an attack). In addition, what event occurs is indicated by the *type* field.: *i.e.*, if label = 0, *normal* event occurs, otherwise it indicates attacks, such as *backdoor*, *DDoS*, *injection*, *password*, *ransomware*, *scanning*, or *XSS*.

Table 1. Train_Test_datasets sensors ranges.

Sensor	Measure	Values
Fridge	Fridge temperature	[1.0; 14.0]
	Temp condition	{'high', 'low'}
Garage Door	Door state	{'open', 'closed' }
	Sphone signal	{'true', 'false', 0, 1 }
GPS Tracker	Latitude	[0.00; 550.00]
	Longitude	[10.00; 556.00]
Modbus	FC1 Read Input Register	$\mathbb{N} \cap [0; 65535]$
	FC2 Read Discrete Value	$\mathbb{N} \cap [0; 65535]$
	FC3 Read Holding Register	$\mathbb{N} \cap [0; 65535]$
	FC4 Read Coil	$\mathbb{N} \cap [0; 65535]$
Motion Light	Motion status	0, 1
	Light status	{'on', 'off'}
Thermostat	Current temperature	[25.00; 35.00]
	Thermostat status	{0, 1}
Weather	Temperature	[20.00; 50.00]
	Pressure	[–12.00; 12.50]
	Humidity	[0.20; 100.00]

2.2 Related Work

Anomaly detection and analysis of unexpected states in IoT devices is a critical research area for various domains, including assisted living, automated operations in industries, and healthcare [9,10]. Several anomaly detection techniques have been proposed to identify abnormalities, outliers, and specific events, by exploiting both time-series and non-temporal data from various domains that can be applied in IoT-specific scenarios [11].

The use of Machine Learning (ML) techniques to create Intrusion Detection Systems is a widely discussed topic in the literature [12–16]. The typical approach involves a thorough analysis of the dataset to identify functional dependencies. Subsequently, the dataset is preprocessed to extract features before applying an artificial intelligence (AI) model. The datasets used in the process are constructed using physical and virtual devices that communicate over various network configurations, including peer-to-peer and client-server setups, such as web and cloud services, or client-broker setups, such as MQTT servers, Node-Red, and others. These datasets are specifically designed for IoT applications.

All of the previously cited documents rely on the use of energy-consuming terminals to perform the role of IDS. However, a more interesting alternative is the possibility of utilizing IoT devices that are already integrated into the system or dedicated IoT devices for this purpose. These devices are significantly more energy-efficient. Efforts in this direction are still in development in the literature. Tekin et *al*. [17] discussed the timing, consumption, and accuracy of various AI models based on algorithms, such as Logistic Regression, Random Forest, K-Nearest Neighbor, Decision Tree, Naive Bayes, or Multi-Layer Perceptron. However, it does not provide a clear definition of the structure of each model and the memory used, making replication of the results challenging. The findings indicate the possibility of a hybrid training approach that combines network and edge computing. However, the challenge of weight sharing needs to be addressed [4].

3 Dataset Processing

To detect attacks against an IoT system, as well as malicious behavior of the monitored smart objects, the TinyIDS exploits sensor readings provided by the observed IoT devices (*e.g.*, telemetry data transmitted by drones) or data directly observed in the field (*e.g.*, the change of path by a compromised drone). Therefore, to evaluate the presented solution TON_IoT Dataset has been exploited [7]. The data processing starts converting the seven IoT datasets into just one big dataset consisting of all readings of all sensors. The procedure consists of a few subsequential steps:

– *Concatenation*: Takes the source CSV files from seven sensors as input and outputs a merge of the records into a single DataFrame object [18] whose columns contain all features.

- *Assembling*: Takes a data frame object and collapses the rows which are of identical timestamps, only taking account of the last one, to reduce the number of records and create the corresponding DataFrame object.
- *Division*: Takes as input a series of samples and outputs a series divided into blocks of the given size. As a general rule, this process intends to exploit the development of sensor measurements over time and therefore should preserve as much temporal continuity as possible. As in every valid machine learning application, training and test datasets must be disjoint. The TON_IoT dataset does not yet contain separate training and test sets, so it is broken down into pieces of a parameterizable dimension. It is more likely that a clear picture of system development and attack patterns will be obtained if the number of subsequent measurements contained in a single piece of data is higher.
- *Splitting*: Specifies two separate datasets, training and test sets, composed of records from these blocks, which are the blocks of packaged rows that were created in the previous step as input. The splitting step decides whether a segment of the assembled dataset records will be part of the training or the testing dataset. A random seed is assigned for reproducing the uniform distribution given the percentage driving the partitioning.
- *Attribution*: Gets a series of records and provides the same records, which are appropriately modified by filling in the missing values. The sensors in the Train_Test_datasets have different sampling and periods of working. To maintain the maximum extent of detection, it is necessary to keep attention on all measurements that have at least one sensor transmitting. When a sensor does not output any data, "missing values" can be found.

It is worth noting that the step *division* explained above may be better understood taking into consideration the Fig. 1.

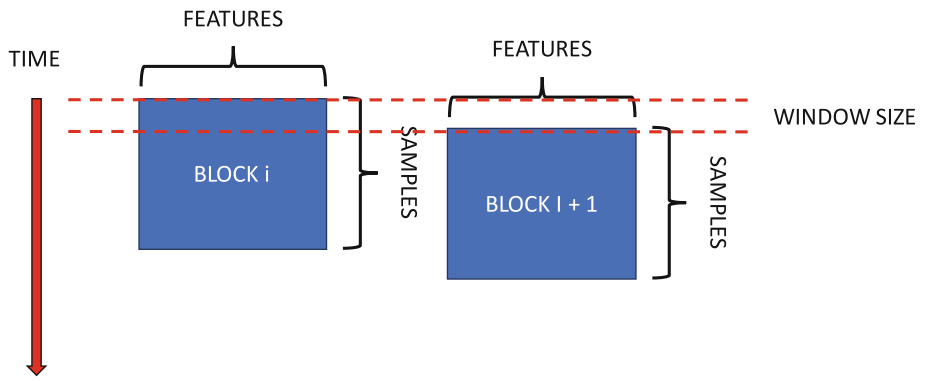

Fig. 1. Dataset blocks.

It is important to note that each block, either included in the training dataset or test dataset, contains more than one sample. To allow time-based learning and

to make the detection system more robust for the measurement of floods, the *division* step is performed. In subsequent steps, a fixed number of records for machine learning will be used to support training based on the past state. The machine can take into account the system's past state of affairs. The selection of the fixed number of records for machine learning supports an analysis based on past states in later phases. The previous state of the system can therefore be taken into account by the machine.

4 Model Implementation

The implementation of the neural network is based on the multilayer perceptron (MLP), also known as a feedforward network. It consists of 510 input neurons, which correspond to a sliding window of 30-time instants, 32 neurons for a 32-layer of fully connected neurons with a nonlinear activation function, represented by the RELU function in our case, and finally, 8 output neurons. These output neurons allow for the identification of either the normal state or the 7 possible threats.

The network was trained using the technique proposed in article [6], which utilizes the Genann library. This C library has been optimized for microcontrollers (including Arduino, Raspberry, and Espressif boards) to perform on-board training and inference. Moreover, it has been updated using float data type instead of double in order to reduce memory usage without compromising the performance of the model [19]. Moreover, the library has been improved with proper metrics (which can be stored in a CSV file) to assess the effectiveness of the newral network. Finally, it supports the export and import of the model's weights for a possible federated learning implementation.

This library allows for the selection of network weights based on the intended device usage context. The weights can be deployed from a previously trained model on more powerful machines or trained live directly on the device. However, this article uses a hybrid approach where the network is pre-trained on a significant dataset and then fine-tuned through on-board training. The training procedure pseudocode is shown in Algorithm 1. It consists of an initial part where all the weights of the neural are initialized to random values. After, it is possible to find out two nested loops, the outer loop goes through the epochs until the last epoch is reached. The inner loop is broken down by the batches of training data collected. It is necessary to step in a different direction from the cost curve of the whole training data set, so as to reduce loss function. Therefore, the gradient approach is sometimes referred to as a batch descent.

By using a dataset with millions of data points, that is not uncommon in many applications for machine learning, in such scenarios carrying out *Batch Gradient Descent* (BGD) can be pretty high in terms of computational costs because every time the algorithm steps up to the Global Minimum, it requires a reevaluation of your entire training database. The *Stochastic Gradient Descent* (SGD) is considered a valid alternative to the BGD algorithm. Rather than update the weights according to the number of errors that have accumulated in all training examples for N, x_i:

$$\nabla \mathcal{L}_i = \sum_{i=1}^{N} \big[y_i - y_{pred}(x_i) \big] x_i,$$

$$\theta_i^{k+1} \leftarrow \theta_i^k - \eta \nabla \mathcal{L}_i.$$

SGD corrects the weights for each training example in a gradual way. Although SGD may be considered an approximation of the gradient descent, as a result of increasing frequency of weight changes, it usually approaches convergence much more quickly.

The Mini-Batch Learning (MBL) algorithm may be considered a point of equilibrium between BGD and SGD algorithms. The application of BGD to a smaller portion of the original training data, *e.g.*, n training examples at a time, can explain MBL. The advantage of minibatches compared to BGD is that convergence can be achieved more quickly due to a higher frequency of weight changes.

Algorithm 1. Training procedure.

Initialization of all weights of neural network;
$\theta \leftarrow \mathbb{R}$ ▷ ML weights are initialized with floating point
for $epoch = 1, 2, \ldots, MAX_EPOCHS$ **do**
  for $batch_data = 1, 2, \ldots, N$ **do**
    FEEDFORWARD PASS
    $\hat{y} \leftarrow model(\mathbf{x}_{batch}, \theta^{epoch})$
    BACKWARD PASS
    $\forall \theta_i : \nabla \mathcal{L}_i \leftarrow \frac{\partial \mathcal{L}}{\partial \theta_i}$
    $\theta_i^{batch} \leftarrow \theta_i^{batch-1} - \eta \nabla \mathcal{L}_i$
  end for
end for

The GitHub project of an implementation of the TinyIDS is available at: *https* : *//github.com/iotresearchunisa/TinyIDS*.

5 Measurements and Results

Following the creation of the model, the experimental phase was carried out on ESP8266, ESP32, and Arduino Portenta H7 devices. The PlatformIO development platform was used to compile the code. It is an integrated development environment (IDE) that provides all the necessary libraries to create code for embedded products [20]. After compilation, the platform performed static code analysis and provided the amount of memory required to run the program.

Table 2 presents the results for ESP32 and Arduino Portenta H7 devices, but not for the ESP8266. The ESP8266 was unable to accommodate all network nodes due to its limited memory capacity of only 32kb of RAM.

Table 2. Memory consumption measurement (PlatformIO building tools).

DEVICES	Memory consumption (kb)
ESP32	114.492
Arduino Portenta H7	157.304

To avoid any interference with the algorithm, a benchmarking station was used, which consisted of an Arduino UNO R3 and an INA219 sensor capable of measuring voltage, power, and current. The benchmarking station provides data in batches to the device under analysis and measures the execution time and current required to perform a single training or inference step. The connection between the benchmarking station and the device under analysis was established via serial communication. Table 3 presents the average of the results obtained.

Table 3. Time and electric current measurement.

Devices	Training		Inference	
	Time (ms)	Current (mA)	Time (ms)	Current (mA)
ESP8266	Failed			
ESP32	2832	71.16	2482	71.23
Arduino Portenta H7	362	216.15	351	223.60

Lastly, a consideration can be made about the accuracy and loss results shown in Figs. 2 and 3 respectively. As it is possible to note, the accuracy curve reaches the convergence quite rapidly. The most important thing is that the curve is completely free from oscillations. Analogously, also in the loss curve oscillations are absent. As the final result, the neural network model provided an accuracy on the test dataset of 84.03

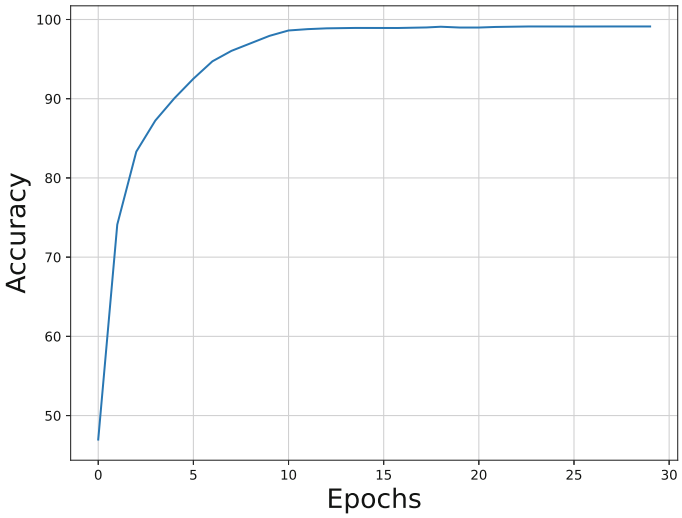

Fig. 2. Training accuracy.

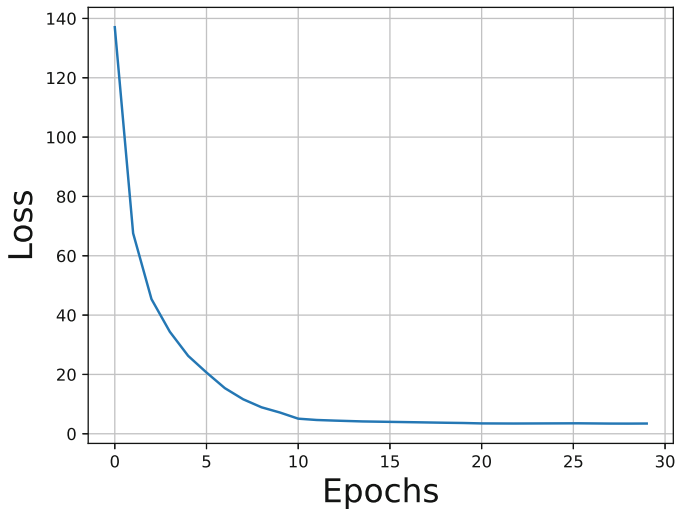

Fig. 3. Training loss.

6 Conclusion

In this paper, an IDS based on Tiny deep learning model is proposed to detect attacks against IoT systems. The presented analysis is performed by using a feedforward network as intrusion detection implementation, which performs its task on the MCU with a limited availability of resources, and the TON_IoT dataset for performance evaluation.

In future works, it is planned to produce a new dataset generated by the physical implementation of more complex and realistic IoT scenarios, as well as increase the detection accuracy of the model by the generated datasets, and the use of the Federated Learning paradigm.

Acknowledgments. The work is part of the research activities realized within the projects Federated Learning for Generative Emulation of Advanced Persistent Threats (FLEGREA), CUP. E53D23007950001, Bando PRIN 2022, as well as SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

References

1. Akram, J., Anaissi, A., Othman, W., Alabdulatif, A., Akram, A.: Dronessl: self-supervised multimodal anomaly detection in internet of drone things. IEEE Trans. Consum. Electron. **70**, 4287–4298 (2024)
2. Pietrantuono, R., Ficco, M., Palmieri, F.: Survivability analysis of IoT systems under resource exhausting attacks. IEEE Trans. Inf. Forensics Secur. **18**, 3277–3288 (2023)
3. Gerodimos, A., Maglaras, L., Ferrag, M.A., Ayres, N., Kantzavelou, I.: IoT: communication protocols and security threats. Internet Things Cyber-Phys. Syst. **3**, 1–13 (2023)
4. Pietrantuono, R., Ficco, M., Palmieri, F.: Testing the resilience of MEC-based IoT applications against resource exhaustion attacks. IEEE Trans. Depend. Secure Comput. **21**(2), 804–818 (2024)
5. Abadade, Y., Temouden, A., Bamoumen, H., Benamar, N., Chtouki, Y., Hafid, A.S.: A comprehensive survey on tinyml. IEEE Access **11**, 96892–96922 (2023)
6. Ficco, M., Guerriero, A., Milite, E., Palmieri, F., Pietrantuono, R., Russo, S.: Federated learning for IoT devices: enhancing TinyML with on-board training. Inf. Fusion **104**, 102189 (2024)
7. Alsaedi, A., Moustafa, N., Tari, Z., Mahmood, A., Anwar, A.: TON_IoT telemetry dataset: a new generation dataset of IoT and IIoT for data-driven intrusion detection systems. IEEE Access **8**, 165130–165150 (2020)
8. Node-RED, a Browser-based flow editing. https://nodered.org. Accessed Sept 2023
9. Chatterjee, A., Ahmed, B.S.: IoT anomaly detection methods and applications: a survey. Internet Things **19**, 100568 (2022)
10. Ukil, A., Bandyoapdhyay, S., Puri, C., Pal, A.: IoT healthcare analytics: the importance of anomaly detection. In: Proceedings of the IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), pp. 994–997 (2016)
11. Fahim, M., Sillitti, A.: Anomaly detection, analysis and prediction techniques in iot environment: a systematic literature review. IEEE Access **7**, 81664–81681 (2019)

12. Smiliotopoulos, C., Kambourakis, G., Kolias, C.: Detecting lateral movement: a systematic survey. Heliyon **10**(4), 1–24 (2024)
13. Kodyš, M., Lu, Z., Fok, K.W., Thing, V.L.L.: Intrusion detection in internet of things using convolutional neural networks. In: 2021 in Proceedings of the 18th International Conference on Privacy, Security and Trust (PST), pp. 1–10 (2021)
14. Altunay, H.C., Albayrak, Z.: A hybrid CNN+ LSTM-based intrusion detection system for industrial IoT networks. Eng. Sci. Technol. Int. J. **38**, 101322 (2023)
15. Thein, T.T., Shiraishi, Y., Morii, M.: Personalized federated learning-based intrusion detection system: poisoning attack and defense. Future Gener. Comput. Syst. **153**, 182–192 (2024)
16. Mahadevappa, P., Murugesan, R.K., Al-Amri, R., Thabit, R., Al-Ghushami, A.H., Alkawsi, G.: A secure edge computing model using machine learning and IDS to detect and isolate intruders. MethodsX **12**, 102597 (2024)
17. Tekin, N., Acar, A., Aris, A., Uluagac, A.S., Gungor, V.C.: Energy consumption of on-device machine learning models for IoT intrusion detection. Internet Things **21**, 100670 (2023)
18. Pandas, a flexible data analysis and manipulation library for Python. https://github.com/pandas-dev/pandas. Accessed Oct 2023
19. Nadalini, D., Rusci, M., Benini, L., Conti, F.: Reduced precision floating-point optimization for Deep Neural Network On-Device Learning on microcontrollers. Future Gener. Comput. Syst. **149**, 212–226 (2023)
20. PlatformIO IDE, a toolset for embedded C/C++ development. https://platformio.org/. Accessed Nov 2023