Lightweight Machine Learning Models for Intrusion Detection on IoT Devices

Jonathan Lundqvist, Anel Hadzic, Torstein Mo Kirkeluten, Håkon Pedersen, Jacob Holth, Magnus H. Johansson, and Moritz P. N. Halkjelsvik

Faculty of Computer Science, Engineering and Economics,
Østfold University College, Halden, Norway
{jrlundqv, anelh, torstemk, haakonp, jacobh, mejohan, mphalkje}@hiof.no

Abstract. The Internet of Things (IoT) continues to expand rapidly, yet IoT devices often lack on-device Intrusion Detection Systems (IDS) due to strict CPU, RAM, and memory limitations. While prior research has explored machine learning-based IDS, relatively few studies have evaluated models under realistic resource constraints. In this paper, we assess the feasibility of lightweight and efficient machine learning models for multi-class intrusion detection using the ToN_IoT dataset. We compare a Decision Tree (DT), Light Gradient Boosting Machine (LightGBM), a Feedforward Neural Network (FNN), and a TinyML-optimized FNN. Our results show that the TinyML FNN provides the best balance between accuracy and deployability, achieving an F1-Score of 0.976, an inference throughput of ≈120,000 packets per second, and a model size of only 31 KB. These findings demonstrate that TinyML-optimized neural networks are strong candidates for practical on-device intrusion detection on highly resource-constrained IoT devices.

Keywords: Lightweight · Cybersecurity · IoT · TinyML · IDS · ToN_IoT

1 Introduction

The rapid growth of the Internet of Things (IoT) technology has drastically increased the number of devices that are connected to the Internet. The International Data Corporation (IDC) estimates that there will be 55.7 billion connected IoT devices (or "things") by 2025 [21]. Although these devices bring great innovation and convenience, they also pose significant cybersecurity risks. One prominent example is the rise of large-scale botnets that exploit vulnerable IoT devices to carry out distributed denial-of-service (DDoS) attacks [12].

Traditional Intrusion Detection Systems (IDS) and Intrusion Preventions Systems (IPS) are not well suited to the resource constraints of IoT devices. They are typically deployed centrally at routers or firewalls and rely on signature-based detection or computationally intensive machine learning (ML) models. However, many IoT devices are directly connected to the internet either over mobile (GSM) networks (4G/5G) or non-Ethernet/Wi-Fi networks, bypassing the traditional central placed IDS solutions. Such devices often have only rudimentary security

features, lack fully functional IDS capabilities, and rarely receive timely software updates or patches. This leaves them highly vulnerable to attacks.

This presents a clear challenge: how can intrusion detection be performed directly on IoT devices that run on severely constrained hardware, such as microcontrollers and ultra low-power IoT devices? Without on-device IDS solutions, many resource-constrained IoT devices will remain vulnerable entry points in global networks.

Previous studies on machine learning models for IDS on IoT devices have mainly focused on classification performance. In contrast, they often overlook resource constraints such as model size, computational load and throughput efficiency. If a device must use most of its resources to run intrusion detection, it may struggle to perform its primary functions like controlling sensors or handling application tasks. Balancing detection performance with deployability on constrained hardware is therefore essential for real-world use.

This gap in the literature has motivated this paper. We aim to build and evaluate a model that is lightweight enough to be deployed to current resource-constraint IoT devices while achieving strong multi-class intrusion detection performance. We evaluate performance metrics such as accuracy, precision and F1-Score. Additionally, we also evaluate model size, CPU frequency (in MHz), and packet throughput (in packets per second, pps) to assess efficiency under resource constraints. Experiments were done on a Raspberry Pi 4 device, where we also reduced the number of active cores and clock speeds (MHz) to mimic other resource-constrained IoT devices (see Table 5 in Section III Methods). Using the ToN_IoT dataset, we build and compare multiple models including a Decision Tree (DT), a Light Gradient Boosting Machine (LightGBM), a Feedforward Neural Network (FNN) and a TinyML-optimized FNN.

Our key contributions are:

1. **Lightweight model:** A TinyML-optimized FNN of 31 KB, 19× smaller than the largest baseline yet with comparable accuracy.
2. **Multi-class detection:** Evaluation of lightweight IDS models on nine attack types from the ToN_IoT dataset.
3. **Hardware validation:** Real-device tests on Raspberry Pi 4, showing ≈120,000 pps throughput under constrained settings.

The remainder of this paper is organized as follows: Section 2 presents an overview of the related work. Section 3 details the dataset, comparable IoT devices, preprocessing, models, model evaluation and performance testing methodology. Section 4 presents the results of the training and the hardware testing. Finally, Section 5 concludes the paper with a discussion of the findings and future work directions.

2 Related Work

2.1 Machine Learning IDS for IoT

Intrusion detection in IoT networks has gained increasing attention, especially with the rise of lightweight machine learning approaches optimized for edge environments. Thakkar and Lohiya [44] provide a comprehensive overview of IDS strategies in IoT, emphasizing the limitations of centralized systems and highlighting the need for resource-efficient, distributed alternatives. Amouri et al. [10] explore this further by implementing a two-stage detection system using Random Forest and linear regression, achieving strong performance in mobile IoT scenarios. Verma and Ranga [46] evaluate several classifiers, including Classification and Regression Trees (CART) and Extreme Gradient Boosting (XGBoost), and examine both detection accuracy and response time on Raspberry Pi hardware. Their results show that certain models are well-suited for on-device intrusion detection in constrained settings. Moustafa et al. [32] used the ToN_IoT dataset, which includes a wide range of realistic attack types and is commonly used for benchmarking ML-based Intrusion Detection Systems (IDS) in IoT research. This dataset provides a comprehensive and diverse set of network traffic data, making it suitable for training and evaluating anomaly detection models.

2.2 IoT and TinyML

IoT environments have traditionally employed relatively lightweight machine learning models such as Decision Trees, Random Forest, and Support Vector Machines because of their simplicity and lower computational complexity than deep neural networks. However, even these models may still require computational resources that exceed what is available in highly constrained IoT devices, especially when real-time inference or ultralow energy consumption is essential [7,13,39]. TinyML addresses this challenge by utilizing model compression, quantization, and pruning techniques to significantly reduce computational and memory requirements, allowing lightweight and efficient models to run directly on highly constrained IoT devices, even those powered by low-energy microcontrollers [24,19,8]. Javed et al. [23] also demonstrated a TinyML-based IDS deployed on a smart thermostat, achieving high accuracy for binary intrusion detection while running locally on-device.

TinyML is widely recognized as a key enabler for low-power, intelligent edge computing, particularly within the context of IoT. Several surveys have outlined the TinyML landscape, identifying key building blocks such as optimized neural network architecture (e.g., MobileNet, SqueezeNet), software frameworks like TensorFlow Lite Micro and uTensor, and hardware platforms including ARM Cortex-M and RISC-V microcontrollers. These components collectively enable on-device inference in energy and memory-constrained environments, making them directly relevant for deploying IDS functionality on low-power IoT devices [8,37]. The literature commonly emphasizes TinyML's potential to reduce latency,

enhance data privacy, and enable offline processing—features that are especially valuable in resource-constrained or intermittently connected environments [15,14].

In addition to theoretical overviews, a growing number of studies address practical implementations of TinyML across various domains. Examples include energy-efficient monitoring systems that utilize lightweight neural networks [16], as well as adaptive inference strategies that dynamically shift computation between the device and the cloud based on available energy [36]. Other works have proposed on-device compression algorithms tailored for IoT scenarios, supporting real-time anomaly detection and efficient bandwidth use [40], and others generally focusing on energy consumption [45,27,43].

A consistent theme across the literature is the pursuit of extreme energy efficiency, making TinyML a compelling solution for always-on, battery-powered, and even battery-less devices operating in real-world environments.

2.3 Limitations and Research Gaps

Despite significant advances, existing research on TinyML-based and machine learning-driven IDS for IoT shows persistent gaps. Many studies rely on synthetic datasets or simplified environments [46,10], limiting real-world applicability, though datasets like ToN_IoT[32] offer some improvement. Most models are statically trained and struggle with evolving threats or dynamic conditions, with only partial solutions proposed through federated learning [29]. Pre-trained models [33,23] are typically designed for uniform hardware setups, overlooking the differences in device capabilities and energy availability found in real environments.

Although energy efficiency is a core theme [16,45,43], its trade-off with detection performance is rarely analysed in depth. Benchmarking practices also lack standardization [28], reducing comparability and reproducibility. Lastly, decentralized and SDN-based systems [9] remain under-explored in terms of scalability and robustness under adversarial conditions. These gaps call for more adaptive, energy-aware, and hardware-agnostic IDS solutions validated in realistic IoT deployments.

3 Methods

The ToN_IoT dataset has received attention from researchers - where classification objectives are focused around best practices to maximize evaluation metrics such as accuracy, precision, recall, and F1-Score. These metrics are important and have even been standardized in ISO/IEC 4213:2022 to ensure comparability across machine learning evaluations. However, these metrics primarily capture the classification ability of a model and do not reflect the challenges of real-world IoT deployments, where factors such as model size, memory usage and inference speed are equally critical.

The goal of the methods described in this section is to find a balance between the classification task and the deployment. It includes dataset selection,

preprocessing techniques, model selection, evaluation criteria and our performance testing methodology. These choices ensure that we develop not only a powerful classifier but also a model that is lightweight, fast, and dependable for resource-constrained devices.

3.1 Dataset ToN_IoT

The ToN_IoT dataset consists of new generations of Industry 4.0/Internet of Things (IoT) and Industrial IoT (IIoT) datasets [32,5]. The testbed was designed to replicate the architectures of IoT systems, giving it an edge with network scenarios that are more representative compared to other available datasets [31]. The dataset was chosen because of its relevance in recent studies on similar topics and its focus on attack types for IoT devices. Although most of them include 300,000 rows of normal type, the one we selected contained only 50,000. With its 211,043 rows and 44 features, it still provides a substantial amount of data to build a model capable of generalizing to different scenarios. We selected the dataset from the `Train_Test_Network_dataset` directory [5]. The dataset includes nine distinct attack type classifications. These are scanning, Distributed Denial of Service (DDoS), Denial of Service (DoS), backdoor, ransomware, Cross-Site Scripting (XSS), password, injection, and Man-in-the-Middle (MITM). Table 1 shows the distribution by category. Certain categories are subject to class imbalances, such as MITM and Normal types.

Table 1. Categories and their corresponding number of rows

Category	Number of Rows
Backdoor	20 000
DDoS	20 000
DoS	20 000
Injection	20 000
MITM	1 043
Normal	50 000
Password	20 000
Ransomware	20 000
Scanning	20 000
XSS	20 000

3.2 IoT Device Considerations

To understand the importance of the size of ML models, we conducted a search for resource-constrained microcontrollers. Many IoT devices are limited in terms of available Flash (program storage), SRAM (runtime memory), and CPU clock speeds. Table 2 shows the devices we found, the first device has 5G capability through a USB interface and an external modem, the next two devices are typical

Wi-Fi IoT devices, and the last one includes a ZigBee IoT device. As shown in the table, such devices typically provide only a few hundred kilobytes of SRAM and up to 1–2 MB of Flash, with processors typically clocked in the lower to mid hundreds of MHz. This means that models exceeding a few hundred kilobytes in size, or requiring continuous high clock speeds to sustain inference, cannot realistically be deployed on such platforms.

Another important aspect of this research included finding whether our model could be compatible with these devices. Libraries such as `emlearn-micropython` [22] and `emlearn` [34] formats the models to be compatible with microcontrollers. Warden et al. [48] demonstrates how Python-made `tflite` models that can be converted to a C file, importing them into C++ environments. These findings indicate that our TinyML FNN model can be deployed to most C, C++ and Python microcontroller environments with consideration to resource capacity.

Table 2. Compatible IoT devices

Device	Flash	SRAM	CPU clock
STM32H747XI [4]	2 MB	864 KB	480, 240 MHz
Arduino Nano ESP32 [1]	16 MB	512 KB	2 x 240 MHz
Raspberry Pi Pico W [3]	2 MB	264 KB	2 x 133 MHz
nRF52840 Express [2]	1 MB	256 KB	1 x 64 MHz

3.3 Data Preprocessing

Despite the dataset being processed by the UNSW Canberra at ADFW [5], additional preprocessing was necessary to ensure consistency and prepare the data for training. The target was set to `type` for multi-classification. Therefore, the following steps were performed:

1. We started with 44 features. The columns `src_ip`, `dst_ip`, and `label` were dropped because they provide unrealistic information about the target and would cause data leakage.
2. Features with more than 65% missing values were removed, eliminating 21 columns.
3. Duplicate rows were eliminated to avoid generalization on repeated data.
4. The target variable (`type`) was label-encoded.
5. The dataset was split into an 80% training set and a 20% test set using stratified sampling to maintain the class distribution.
6. To evaluate the effectiveness of the model on unseen data, a 5-fold cross-validation was performed on the training split before evaluating on the test set.

After splitting, additional preprocessing steps were performed for the neural network. To reduce the impact of extreme values, numerical outliers were capped

using the interquartile range method. Features with skewed distribution were normalized using a logarithmic transformation. All numeric features were standardized, and categorical variables were one-hot encoded, creating 25 dummy features. Features with high correlation (Pearson's $r > 0.90$) were then removed to eliminate redundancy, reducing the feature set to 38 in total.

For the Decision Tree and LightGBM, after the split we only label-encoded the categorical features and trained on the raw numeric values. No other transformations were performed, since tree-based models are capable of handling these variances.

3.4 Models

This subsection presents the machine learning models selected for evaluation. Each was chosen based on its potential balance between classification performance and suitability for deployment on resource-constrained IoT devices.

Decision Tree A Decision Tree classifier was chosen due to its simplicity and interpretability. Its inclusion enables a comparison against more complex and potentially resource-intensive models within the IoT context.

Feedforward Neural Network (FNN) A standard FNN was included to explore the applicability of neural-based methods for intrusion detection in IoT networks. FNNs are capable of modelling non-linear decision boundaries, which is useful for distinguishing between normal and malicious traffic patterns [38]. Prior work has demonstrated the suitability of FNNs for intrusion detection tasks due to their flexibility and ability to generalize across diverse datasets [6]. Including a non-optimized FNN provided a baseline neural architecture for comparison with its TinyML-optimized FNN counterpart.

TinyML-Optimized Feedforward Neural Network To address the constraints of IoT devices, a FNN was optimized using TinyML techniques such as quantization and pruning. TinyML enables the deployment of machine learning models on resource-limited devices by reducing model size and computational demands [36]. This optimization aims to balance classification performance with the possibility of on-device deployment, making it suitable for real-time intrusion detection in IoT environments.

Light Gradient Boosting Machine (LightGBM) LightGBM is a gradient boosting framework known for its efficiency and high performance in classification tasks. It utilizes histogram-based learning and leaf-wise tree growth to improve training speed and memory usage compared to traditional gradient boosting methods [25]. While primarily designed for high-performance environments, its inclusion allows for a benchmark comparison with lighter models in terms of both accuracy and resource demands.

3.5 Model Evaluation

The evaluation metrics played a vital role in understanding the overall classification performance of the models. We used standard classification metrics such as *accuracy*, *precision*, and the *F1-Score*. The dataset itself contained imbalanced class distributions, specifically for the normal and MITM type. Relying on the accuracy alone could be misleading due to the accuracy paradox. Including additional metrics helped to gather insight into the model performance and the balance of the results. In the context of attack classifications, these metrics are explained as follows:

- *Accuracy*: Measurement of how often the model correctly identified an instance.
- *Precision*: Measurement of how many times the model correctly classifies a predicted attack as an actual attack. This is relevant because of the imbalanced normal category, we want to reduce false positives (not often classifying normal as attack)
- *F1-Score*: Measurement combining recall and precision. The score is often used to balance recall and precision and is especially useful to evaluate uneven class distributions along with false positives and false negatives. Due to the imbalanced nature of the dataset, we will be using this measurement as our main evaluation metric.

3.6 Performance testing methodology

The Raspberry Pi 4 was selected as the hardware platform for evaluating our machine learning models. The decision was primarily motivated by the widespread availability and compatibility of Python-based machine learning frameworks on the Linux-based Raspberry Pi OS. This made the process of deployment and testing of our ML models straightforward.

Compared to devices such as microcontrollers and ultra low power edge devices, the Raspberry Pi 4 provides relatively higher computational performance. It features a 1.5 GHz quad-core ARM Cortex-A72 processor and 4 GB RAM[35]. It does not necessarily represent the constraints of the most resource-limited IoT devices. However, we were able to restrict the processor to run at any number of cores and apply hardware limits to clock them as low as 600 MHz, closely mimicking the clock speeds of certain microcontrollers [4,1] (Table 2). We were therefore able to test all of our models under various hardware configurations, as described in Table 3.

Table 3. Overview of test hardware configurations

Active cores	Core clock	Total CPU clock
4	1500 MHz	6000 MHz
1	1500 MHz	1500 MHz
1	600 MHz	600 MHz

This way testing on the Raspberry Pi 4 provided a more realistic context than what would be obtained from evaluations on traditional laptops or desktop computers. This platform allowed us to measure inference performance and resource utilization under conditions that are closer to typical IoT deployment scenarios.

Python environment The performance testing of our machine learning models was implemented using Python version 3.10, relying on several libraries to enable model inference and performance monitoring. Tree models serialized into the `.pkl` file structure were loaded using the `joblib` library. Model inference was then executed using their respective framework; `scikit-learn` for our decision tree model, and `lightgbm` for our gradient-boosted tree ensemble model. Neural network models serialized into the `.tflite` file structure were executed with Tensorflow Lite's runtime interpreter `tflite-runtime`. For real-time monitoring of system resources, `psutil` was used, providing measurements of CPU utilization throughout the testing procedure.

Test procedure The test procedure was designed to realistically assess the inference performance of the deployed ML models under controlled conditions. We divided the experiment into three phases: a pre-inference baseline, the inference execution and a post-inference baseline, as can be seen in Figure 1. This design aimed to capture baseline system load both before and after model inference, providing a clear reference for understanding the impact of the ML model on system resources.

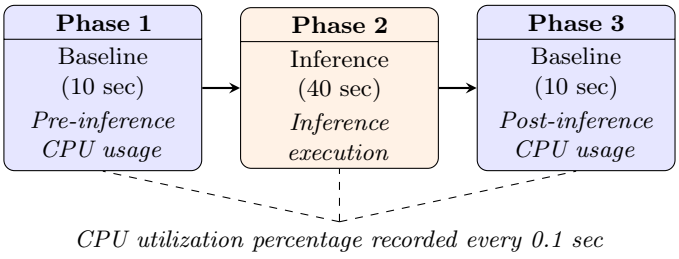

Fig. 1. Overview of the test procedure phases

The inference test was executed either under an uncapped packet rate allowing the maximum throughput achievable by the model, or at a set packet rate simulating a possible operational scenario. The uncapped method aimed to evaluate the maximum practical inference capability of our models on the Raspberry Pi 4 platform. Controlled packet rates tested resource utilization under specific throughput conditions relevant to different deployment scenarios.

Real-time CPU utilization monitoring was measured continuously throughout all phases at fixed intervals. The collected metrics provided quantitative insight into the resource consumption of our ML models operating in these simulated scenarios.

Measuring Model Performance When testing the models for maximum packet rate, the *total packet throughput* (in packets per second, pps) best captures its *performance*. When we perform tests with set packet rates, *CPU utilization* directly reflects its *efficiency*. Because throughput alone speaks only to performance and utilization alone only to efficiency, we introduce a unified metric, *packets per MHz* (packets/MHz), defined as:

$$\text{packets/MHz} \;=\; \frac{\text{Total PPS}}{\big(\text{CPU clock (MHz)}\big) \;\times\; U} \tag{1}$$

where U is the average CPU utilization expressed as a fraction (0–1), and "CPU clock" is the effective clock frequency under load.

This metric grows when uncapped tests yield higher throughput and when capped tests consume less CPU, making it comparable across all configurations. Additionally, given a target device with known clock speeds and assumed utilization, one can estimate its inference throughput as:

$$\widehat{\text{pps}} \;=\; \big(\text{packets/MHz}\big) \;\times\; \big(\text{CPU clock (MHz)}\big) \;\times\; U_{\text{est}} \tag{2}$$

where U_{est} is the projected utilization fraction for that device.

Packet rates In order to understand our model's performance across a broad spectrum of operational scenarios, we have identified three packet-rate cases spanning extreme, high-load, and typical conditions. This lets us validate its capacity limits and real-world applicability across varying traffic intensities.

According to the IMT-2020 Standard, a single 5G carrier is expected to deliver peak downlink speeds of up to 20 Gbps [17]. Translating bit-rates into packet rates depends largely on the size of the payload transmitted. Empirical studies of internet traffic suggest that many typical messages have payload sizes around 500 bytes [26]. At maximum theoretical 5G speeds with a payload of 500 bytes, this equals $\frac{20\text{ Gbps}}{500\text{ bytes}} = 5\,000\,000$ packets per second (pps). This upper-bound scenario allows us to evaluate how the model would perform under extreme network load.

A more realistic yet demanding scenario involves high-resolution video streaming, such as 4K video at 60 frames per second. Such streaming can use approximately 50 Mbps depending on the video codec [42,47]. Given the IPv6 minimum MTU requirement of 1280 bytes [20], this results in a packet rate of $\frac{50\text{ Mbps}}{1280\text{ bytes}} \approx 4882$ pps, which we approximate to 5000 pps for practicality. This scenario could represent IoT applications such as security monitoring using high-resolution video cameras or automated industrial quality inspection systems.

A low-end, typical scenario for IoT devices involves sending telemetry data at regular intervals. This could mean systems such as environmental sensors reporting temperature, humidity, and air quality metrics, or smart-home sensors updating occupancy or power consumption statuses. Such telemetry data generally results in relatively low packet rates, as shown by Sivanathan et al. when they characterized IoT traffic in smart cities and campuses [41]. Therefore, we selected 5 pps as representative of standard IoT operational conditions.

Through this range of test scenarios, as further outlined in Table 4, we aim to assess the feasibility of our ML-based IDS across diverse and realistic IoT deployment conditions.

Table 4. Overview of test scenarios

Type	**Scenario**	**Packets/sec**
Extreme	Close to limits of 5G capacity	5 000 000
High-end	Video streaming at 4K 60FPS	5 000
Typical	Regular telemetry traffic	5

4 Results

4.1 Model evaluation

The results as can be seen in Table 5 include evaluation metrics such as accuracy, F1-Score, precision, and model size. To compare our models performance, other existing works that use the same `train_test_network` dataset has been added. The F1-Score is the main metric for comparison. Regarding the size of the model, we could only compare our own models, since we had the size data for those.

Our Decision Tree achieved the highest F1-Score (0.987) and accuracy (0.987), but its size (593 KB) makes it one of the largest. Although the existing Random Forest from another work [11] had the second highest accuracy (0.981), it was the second worst performer in terms of F1-Score (0.973). The worst performer in terms of F1-Score (0.886) and accuracy (0.963) was the Voting model of another existing work [18]. LightGBM, FNN, and TinyML FNN all achieved similar F1-Scores (0.976). The largest model of these was the LightGBM (599 KB), while the smallest was the TinyML FNN (31 KB). Therefore, the TinyML FNN is the most suitable candidate for deployment on resource-constrained devices.

4.2 Performance test results

Performance testing was conducted on five selected configurations (Table 6) out of the nine possible combinations of hardware configurations (Table 3) and packet rate scenarios (Table 4). Although an "Extreme" 5 000 000 pps scenario was specified to test theoretical 5G limits, no model achieved rates near this threshold, rendering the Extreme and Max throughput runs identical.

Table 5. Multi-Classification Metrics of Our Models vs. Existing Works

Model	Accuracy	F1-Score	Precision	Size
DT	**0.987**	**0.987**	**0.987**	593 KB
LightGBM	0.976	0.976	0.977	599 KB
FNN	0.974	0.976	0.979	57 KB
TinyML FNN	0.974	0.976	0.979	**31 KB**
Existing works				
RandomForest [11]	0.981	0.973	N/R†	N/R†
Voting [18]	0.963	0.886	0.931	N/R†

† "Not reported" (N/R) in the cited works.

We report the max throughput tests on all three hardware configurations (Table 3), to capture each model's performance ceiling. We include two capped-rate tests with packet rates of 5 pps (typical IoT telemetry) and 5 000 pps (High-end video streaming), at a CPU clock of 600 MHz. This clock speed was selected for providing the most granular efficiency data for all models, as all models managed 5 000 pps at this hardware configuration. This selection spans from peak inference speeds to realistic operational loads, enabling comparison via packets per second, CPU utilization, and the unified packets/MHz efficiency metric (Equation (1). Detailed numeric results for these five configurations are provided in Tables 7 to 10, with graphical summaries in Figures 2 and 3.

Table 6. Executed test configurations

Test scenario	Active cores	Core clock
Max throughput	4	1500 MHz
Max throughput	1	1500 MHz
Max throughput	1	600 MHz
5000 pps	1	600 MHz
5 pps	1	600 MHz

Maximum packet rates In our benchmarks, LightGBM stands apart as the only model to leverage true multi-core inference. Thanks to its tree-ensemble design and parallel prediction engine, it is able to distribute inference work across all available CPU cores [30] (see Table 7). This can be advantageous in environments where parallelism is valuable, but comes at the cost of higher total resource use. This is a great feature in high-performance environments, but is less relevant in our resource-aware approach.

By contrast, the Decision Tree stands out as the consistently top-performing model across every scenario (see Figures 2 and 3. Comparing Table 7 and Table 8, the Decision Tree performance is moderately improved in configurations with

multiple cores available, even though the average CPU utilization is only 25 %. There seems to be some process by which the DT enhances its performance slightly in multi-core situations which our test setup is not able to capture.

Our two FNN variants consume nearly identical CPU resources and offer similar throughput. However, the TinyML version achieves this at roughly half the model size, markedly reducing memory footprint without sacrificing performance and efficiency.

Table 7. CPU clock: 4 x 1500 MHz, Max packets per second (PPS)

Model	CPU %	PPS	Packets/MHz
DT	25	612 073	406
LightGBM	99	35 788	6
FNN	26	122 965	80
TinyML FNN	26	123 883	80

Table 8. CPU clock 1 x 1500 MHz, Max packets per second (PPS)

Model	CPU %	PPS	Packets/MHz
DT	99	504 708	341
LightGBM	100	15 600	10
FNN	100	118 377	79
TinyML FNN	99	119 294	80

Table 9. CPU clock 1 x 600 MHz, Max packets per second (PPS)

Model	CPU %	PPS	Packets/MHz
DT	97	213 812	366
LightGBM	100	6 125	10
FNN	99	52 306	88
TinyML FNN	99	52 306	88

Set packet rates Under the lowest packet-rate scenario (5 pps), all models exhibited a baseline CPU utilization of around 3 % (see Table 11), revealing that the overhead is dominated by the Python runtime rather than model inference. At just 5 pps, the time spent inside the model's `predict()` or TensorFlow-Lite interpreter call is vanishingly small compared to the fixed Python loop and system-call overhead. This means that on very low-traffic links, further slimming the model yields diminishing returns unless the entire inference pipeline is optimized for C-level or bare-metal execution.

In the 5 000 pps scenario, the Decision Tree's runtime overhead is nearly indistinguishable from the baseline Python runtime at just 4 % CPU (see Table 10). Meanwhile, both FNN variants demand roughly three times that overhead (12-13 % CPU) to sustain the same throughput.

Table 10. CPU clock 1 x 600 MHz, 5000 packets per second (PPS)

Model	CPU %	PPS	Packets/MHz
DT	4	5 000	199
LightGBM	85	5 000	10
FNN	12	5 000	68
TinyML FNN	13	5 000	66

Table 11. CPU clock 1 x 600 MHz, 5 packets per second (PPS)

Model	CPU %	PPS	Packets/MHz
DT	3.17	5	0.26
LightGBM	3.22	5	0.26
FNN	3.13	5	0.27
TinyML FNN	3.34	5	0.25

4.3 Estimating performance on microcontrollers

Using Equation (2), we can derive an estimate of inference performance on any target device given its CPU clock frequencies and an assumed utilization fraction. However, these estimates carry several important caveats that introduce uncertainty. Our measurements used CPython on a full OS, whereas microcontrollers typically run MicroPython or other minimal interpreters. This change in runtime environment affects performance in unpredictable ways, which we can't account for without further specific runtime tests.

By explicitly acknowledging these factors, our estimates serve as an order-of-magnitude guideline for selecting microcontroller platforms, rather than as exact predictors of performance.

Using Equation (2) and our measured packets/MHz values for each model, we have

$$P_{\mathrm{DT}} = 366, \quad P_{\mathrm{FNN}} = 88,$$

and assuming a conservative utilization fraction $U_{\mathrm{est}} = 0.5$, we calculate, for each device in Table 12,

$$\widehat{\mathrm{pps}}_{\mu} = (\text{packets/MHz}) \times (\text{CPUClock(MHz)}) \times U_{\mathrm{est}}.$$

for each device in Table 2.

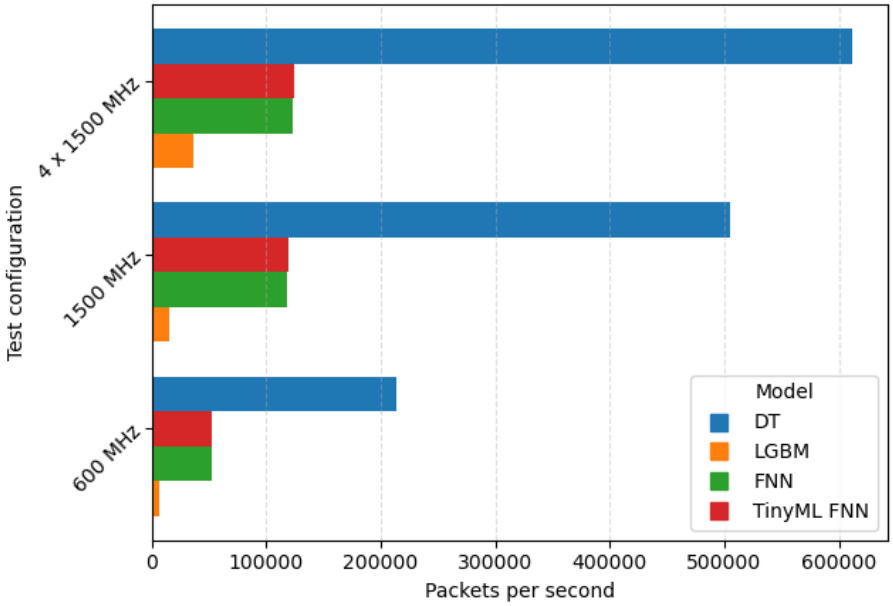

Fig. 2. Packets per second for each model. Showing performance across different CPU-clock configurations.

Table 12. Estimated inference performance of selected IoT devices

Device	$\widehat{\text{pps}}_{\text{DT}}$	$\widehat{\text{pps}}_{\text{FNN}}$
STM32H747XI	131 760	31 680
Arduino Nano ESP32	87 840	21 120
Raspberry Pi Pico W	48 678	11 704
nRF52840 Express	23 424	5 632

Such estimates could be tested for accuracy by converting and deploying models to compatible microcontrollers as previously described in Section III-F.

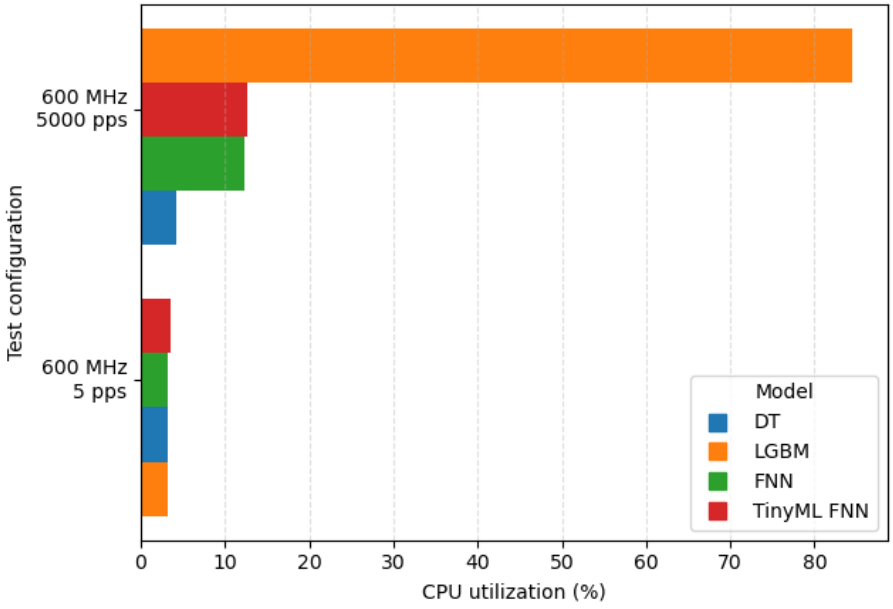

Fig. 3. Inference-phase CPU utilization (%) for each model at set packet rates. Showing efficiency across different packet rate scenarios.

5 Conclusion

In this study, we investigated the feasibility of deploying lightweight machine learning-based IDS directly on IoT hardware. Using the ToN_IoT dataset, we evaluated the performance of our four models on a resource-constrained Raspberry Pi 4 device. The device was in certain test configurations restricted to mimic the processing power of IoT microcontrollers.

Overall, the results indicate a clear trade-off among accuracy, throughput, and memory footprint: the Decision Tree delivered the strongest predictive and throughput performance but exceeded typical memory budgets; LightGBM was accurate yet least efficient and similarly large; and the TinyML FNN provided the most deployable balance for highly constrained devices. These findings suggest that TinyML-style neural networks are promising candidates for on-device intrusion detection in practice.

Our Decision Tree achieved the highest F1-Score (0.987) and accuracy (0.987), but its size (593 KB) makes it one of the largest. Looking at the compatible IoT devices in Table 2, this size is problematic. Three out of four devices have only 256–512 KB of SRAM, which means that the model cannot be deployed on them. Even on the device with 864 KB SRAM, deployment would leave very limited space for firmware and the runtime of other tasks. LightGBM had a formidable F1-Score (0.976), but as expected was the least efficient in our performance

test tallying 10 packets/MHz. Along with its size of 599 KB, its usage is not practical for IoT settings. The TinyML FNN offered the most balanced trade-off, with a competitive F1-Score (0.976), and practical efficiency of 88 packets/MHz. In addition to being the smallest size (31 KB), it indicates its compatibility with low memory devices. With all these metrics in mind, we can conclude that the TinyML model is the most suitable for performing intrusion detection on highly resource-constrained devices. Our results demonstrate the potential of lightweight machine learning algorithms to support practical intrusion detection in resource-constrained environments, a topic that has not been explored in the literature.

Future work could involve tests on microcontroller devices with stricter resource limitations than the Raspberry Pi 4, as it is comparatively quite powerful. Evaluating performance on live network traffic would be an essential next step to understanding performance under realistic conditions. Another important direction is evaluating the models on a broader range of hardware platforms to assess deployment generalizability across the IoT ecosystem.

References

1. Arduino® Nano ESP32, https://store.arduino.cc/products/nano-esp32
2. nRF52840 - Bluetooth 5.3 SoC - nordicsemi.com, https://www.nordicsemi.com/Products/nRF52840
3. Raspberry Pi Pico W • RaspberryPi.dk, https://raspberrypi.dk/en/product/raspberry-pi-pico-w/
4. STM32H747XI - High-performance and DSP with DP-FPU, Arm Cortex-M7 + Cortex-M4 MCU - STMicroelectronics., https://www.st.com/en/microcontrollers-microprocessors/stm32h747xi
5. a ADFA, U.C.: The TON_iot Datasets (Jun 2021), https://research.unsw.edu.au/projects/toniot-datasets
6. Akuthota, U.C., Bhargava, L.: Evaluation of Machine Learning Models for Intrusion Detection with the UNSW-NB15 Dataset. In: 2023 IEEE Silchar Subsection Conference (SILCON). pp. 1–5. IEEE (Nov 2023). https://doi.org/10.1109/SILCON59133.2023.10404204
7. Alabdulkarim, A., Al-Rodhaan, M., Ma, T., Tian, Y.: PPSDT: A Novel Privacy-Preserving Single Decision Tree Algorithm for Clinical Decision-Support Systems Using IoT Devices. Sensors **19**(1), 142 (Jan 2019). https://doi.org/10.3390/s19010142, https://www.mdpi.com/1424-8220/19/1/142
8. Alajlan, N.N., Ibrahim, D.M.: TinyML: Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications. Micromachines **13**(6), 22 (May 2022). https://doi.org/10.3390/mi13060851, https://www.mdpi.com/2072-666X/13/6/851
9. Alsaleh, S., Menai, M.E.B., Al-Ahmadi, S.: A Heterogeneity-Aware Semi-Decentralized Model for a Lightweight Intrusion Detection System for IoT Networks Based on Federated Learning and BiLSTM. Sensors (Basel, Switzerland) **25**(4), 1039 (Feb 2025). https://doi.org/10.3390/s25041039, https://www.mdpi.com/1424-8220/25/4/1039
10. Amouri, A., Alaparthy, V.T., Morgera, S.D.: A Machine Learning Based Intrusion Detection System for Mobile Internet of Things. Sensors **20**(2), 461 (Jan 2020).

https://doi.org/10.3390/s20020461, https://www.mdpi.com/1424-8220/20/2/461, number: 2 Publisher: Multidisciplinary Digital Publishing Institute
11. Booij, T.M., Chiscop, I., Meeuwissen, E., Moustafa, N., Hartog, F.T.H.d.: ToN_iot: The Role of Heterogeneity and the Need for Standardization of Features and Attack Types in IoT Network Intrusion Data Sets. IEEE Internet of Things Journal **9**(1), 485–496 (Jan 2022). https://doi.org/10.1109/JIOT.2021.3085194, https://ieeexplore.ieee.org/abstract/document/9444348
12. Cloudflare: How can IoT devices be used in cyber attacks, https://www.cloudflare.com/learning/ddos/glossary/internet-of-things-iot/
13. Dangore, M., Bhaturkar, D., Bhale, K.M., Jadhav, H.M., Borate, V.K., Mali, Y.K.: Applying Random Forest for IoT Systems in Industrial Environments. In: 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). pp. 1–7. IEEE, Kamand, India (Jun 2024). https://doi.org/10.1109/ICCCNT61001.2024.10725751, https://ieeexplore.ieee.org/document/10725751/
14. Dutta, D.L., Bharali, S.: TinyML Meets IoT: A Comprehensive Survey. Internet of Things **16**, 100461 (Dec 2021). https://doi.org/10.1016/j.iot.2021.100461, https://linkinghub.elsevier.com/retrieve/pii/S2542660521001025
15. Elhanashi, A., Dini, P., Saponara, S., Zheng, Q.: Advancements in TinyML: Applications, Limitations, and Impact on IoT Devices. Electronics **13**(17), 3562 (Sep 2024). https://doi.org/10.3390/electronics13173562, https://www.mdpi.com/2079-9292/13/17/3562
16. Giordano, M., Baumann, N., Crabolu, M., Fischer, R., Bellusci, G., Magno, M.: Design and Performance Evaluation of an Ultralow-Power Smart IoT Device With Embedded TinyML for Asset Activity Monitoring. IEEE Transactions on Instrumentation and Measurement **71**, 1–11 (2022). https://doi.org/10.1109/TIM.2022.3165816, https://ieeexplore.ieee.org/document/9758676/
17. Gosh, I.: Making Smart Grid more Smart With 5G Communication - IEEE Smart Grid (Apr 2021), https://smartgrid.ieee.org/bulletins/april-2021/making-smart-grid-more-smart-with-5g-communication
18. Hajla, S.E., Ennaji, E.M., Maleh, Y., Mounir, S.: Enhancing IoT network defense: advanced intrusion detection via ensemble learning techniques. Indonesian Journal of Electrical Engineering and Computer Science **35**(3), 2010–2020 (Sep 2024). https://doi.org/10.11591/ijeecs.v35.i3.pp2010-2020, http://doi.org/10.11591/ijeecs.v35.i3.pp2010-2020, number: 3
19. Han, L., Xiao, Z., Li, Z.: DTMM: Deploying TinyML Models on Extremely Weak IoT Devices with Pruning (Jan 2024). https://doi.org/10.48550/arXiv.2401.09068, http://arxiv.org/abs/2401.09068, arXiv:2401.09068 [cs]
20. Hinden, B., Deering, S.E.: Internet Protocol, Version 6 (IPv6) Specification. Request for Comments RFC 2460, Internet Engineering Task Force (Dec 1998). https://doi.org/10.17487/RFC2460, https://datatracker.ietf.org/doc/rfc2460, num Pages: 39
21. Hojlo, J.: Future of Industry Ecosystems: Shared Insights & Data (2021), https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/
22. J, N.: emlearn-micropython: Efficient Machine Learning engine for MicroPython, https://zenodo.org/records/14585675
23. Javed, A., Ehtsham, A., Jawad, M., Awais, M.N., Qureshi, A.u.H., Larijani, H.: Implementation of Lightweight Machine Learning-Based Intrusion Detection System on IoT Devices of Smart Homes. Future Internet **16**(6), 200 (Jun 2024). https://doi.org/10.3390/fi16060200, https://www.mdpi.com/1999-5903/16/6/200

24. Kudavelly, S.R., Malavika: TinyML Optimization Approaches for Deployment on Edge Devices (2024), https://www.cyient.com/whitepaper/tiny-ml-optimization
25. Li, J., Othman, M.S., Chen, H., Yusuf, L.M.: Optimizing IoT intrusion detection system: feature selection versus feature extraction in machine learning. Journal of Big Data **11**(1), 36 (Feb 2024). https://doi.org/10.1186/s40537-024-00892-y, https://doi.org/10.1186/s40537-024-00892-y
26. Liu, F., Wu, X., Li, W., Liu, X.: The packet size distribution patterns of the typical Internet applications. In: 2012 3rd IEEE International Conference on Network Infrastructure and Digital Content. pp. 325–332 (Sep 2012). https://doi.org/10.1109/ICNIDC.2012.6418769, https://ieeexplore.ieee.org/document/6418769, iSSN: 2374-0272
27. Liu, R., Xie, M., Liu, A., Song, H.: Joint Optimization Risk Factor and Energy Consumption in IoT Networks With TinyML-Enabled Internet of UAVs. IEEE Internet of Things Journal **11**(12), 20983–20994 (Jun 2024). https://doi.org/10.1109/JIOT.2023.3348837, https://ieeexplore.ieee.org/document/10379459/
28. Mallidi, S.K.R., Ramisetty, R.R.: Optimizing Intrusion Detection for IoT: A Systematic Review of Machine Learning and Deep Learning Approaches With Feature Selection and Data Balancing. WIREs Data Mining and Knowledge Discovery **15**(2), e70008 (2025). https://doi.org/10.1002/widm.70008, https://www.preprints.org/manuscript/202503.0706/v1
29. Mani, P., .D, A., K, P., S, S.: A Lightweight and Federated Machine Learning Based Intrusion Detection System for Multi Attack Detection in IoT Networks. Journal of Machine and Computing pp. 421–429 (Jan 2025). https://doi.org/10.53759/7669/jmc202505033, http://dx.doi.org/10.53759/7669/jmc202505033
30. Microsoft: Parameters Tuning — LightGBM 4.6.0.99 documentation, https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html
31. Moustafa, N.: A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_iot datasets. Sustainable Cities and Society **72**, 102994 (Sep 2021). https://doi.org/10.1016/j.scs.2021.102994, https://www.sciencedirect.com/science/article/pii/S2210670721002808
32. Moustafa, N., Keshky, M., Debiez, E., Janicke, H.: Federated TON_iot Windows Datasets for Evaluating AI-Based Security Applications. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 848–855. IEEE, Guangzhou, China (Dec 2020). https://doi.org/10.1109/TrustCom50675.2020.00114, https://ieeexplore.ieee.org/document/9343133/
33. Murthy, A., Asghar, M.R., Tu, W.: A lightweight Intrusion Detection for Internet of Things-based smart buildings. SECURITY AND PRIVACY **7**(4), e386 (2024). https://doi.org/10.1002/spy2.386, https://onlinelibrary.wiley.com/doi/abs/10.1002/spy2.386
34. Nordby, J.: emlearn: Machine Learning inference engine for Microcontrollers and Embedded Devices (Mar 2019). https://doi.org/10.5281/zenodo.2589394, https://zenodo.org/records/2589394
35. Pi, R.: Processors - Raspberry Pi Documentation (2025), https://www.raspberrypi.com/documentation/computers/processors.html
36. Sabovic, A., Aernouts, M., Subotic, D., Fontaine, J., Poorter, E.D., Famaey, J.: Towards energy-aware tinyML on battery-less IoT devices. Internet of Things **22**, 100736 (2023). https://doi.org/https://doi.org/10.1016/j.iot.2023.100736, https://www.sciencedirect.com/science/article/pii/S2542660523000598

37. Schizas, N., Karras, A., Karras, C., Sioutas, S.: TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review. Future Internet **14**(12), 363 (Dec 2022). https://doi.org/10.3390/fi14120363, https://www.mdpi.com/1999-5903/14/12/363
38. Shaker, B.N., Al-Musawi, B.Q., Hassan, M.F.: A Comparative Study of IDS-Based Deep Learning Models for IoT Network. In: Proceedings of the 2023 International Conference on Advances in Artificial Intelligence and Applications. pp. 15–21. AAIA '23, Association for Computing Machinery, New York, NY, USA (Jan 2024). https://doi.org/10.1145/3603273.3635058, https://dl.acm.org/doi/10.1145/3603273.3635058
39. Shen, M., Tang, X., Zhu, L., Du, X., Guizani, M.: Privacy-Preserving Support Vector Machine Training Over Blockchain-Based Encrypted IoT Data in Smart Cities. IEEE Internet of Things Journal **6**(5), 7702–7712 (Oct 2019). https://doi.org/10.1109/JIOT.2019.2901840, https://ieeexplore.ieee.org/document/8653362/
40. Signoretti, G., Silva, M., Andrade, P., Silva, I., Sisinni, E., Ferrari, P.: An Evolving TinyML Compression Algorithm for IoT Environments Based on Data Eccentricity. Sensors **21**(12), 4153 (Jun 2021). https://doi.org/10.3390/s21124153, https://www.mdpi.com/1424-8220/21/12/4153
41. Sivanathan, A., Sherratt, D., Gharakheili, H.H., Radford, A., Wijenayake, C., Vishwanath, A., Sivaraman, V.: Characterizing and classifying IoT traffic in smart cities and campuses. In: 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 559–564. IEEE, Atlanta, GA (May 2017). https://doi.org/10.1109/INFCOMW.2017.8116438, http://ieeexplore.ieee.org/document/8116438/
42. Sullivan, G.J., Ohm, J.R., Han, W.J., Wiegand, T.: Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Transactions on Circuits and Systems for Video Technology **22**(12), 1649–1668 (Dec 2012). https://doi.org/10.1109/TCSVT.2012.2221191, http://ieeexplore.ieee.org/document/6316136/
43. 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 of Things **21**, 100670 (Apr 2023). https://doi.org/10.1016/j.iot.2022.100670, https://linkinghub.elsevier.com/retrieve/pii/S2542660522001512
44. Thakkar, A., Lohiya, R.: A Review on Machine Learning and Deep Learning Perspectives of IDS for IoT: Recent Updates, Security Issues, and Challenges. Archives of Computational Methods in Engineering **28**(4), 3211–3243 (Jun 2021). https://doi.org/10.1007/s11831-020-09496-0, https://doi.org/10.1007/s11831-020-09496-0
45. Thuppari, C., Jannu, S., Edla, D.R., Vidyarthi, A., Agarwal, K.K., Alkhayyat, A.: Energy-Aware Compression and Consumption Algorithms for Efficient TinyML Model Using Aquila Optimization in Industrial IoT. IEEE Transactions on Consumer Electronics pp. 1–1 (2024). https://doi.org/10.1109/TCE.2024.3475393, https://ieeexplore.ieee.org/document/10707135/
46. Verma, A., Ranga, V.: Machine Learning Based Intrusion Detection Systems for IoT Applications. Wireless Personal Communications **111**(4), 2287–2310 (Apr 2020). https://doi.org/10.1007/s11277-019-06986-8, https://doi.org/10.1007/s11277-019-06986-8
47. Wang, M., Xu, C., Jia, S., Muntean, G.M.: Video streaming distribution over mobile Internet: a survey. Frontiers of Computer Science **12**(6), 1039–1059 (Dec 2018). https://doi.org/10.1007/s11704-018-7153-6, https://doi.org/10.1007/s11704-018-7153-6

48. Warden, P., Situnayake, D.: TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. In: TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers, pp. 69–70. 1 edn. (2019)