

Article

Lightweight Machine Learning Intrusion Detection for IoT/IoT Networks: Quantisation Strategies and Physical Deployment on Resource-Constrained Microcontrollers

Emanuele Pio De Bernardis ¹, Oleksandr Kuznetsov ^{1,2,*}, Marco Arnesano ¹, Polatova Zhansaya ³ and Madina Sydykova ³

¹ Department of Computer Science and Engineering, Università Telematica eCampus, 22060 Novedrate, Italy; emanuelepio.debernardis@gmail.com (E.P.D.B.); marco.arnesano@unicampus.it (M.A.)

² Department of Intelligent Software Systems and Technologies, School of Computer Science and Artificial Intelligence, V.N. Karazin Kharkiv National University, 61022 Kharkiv, Ukraine

³ Department of Computer Engineering, Almaty Technological University, Almaty 050012, Kazakhstan; zh.polatova@atu.edu.kz (P.Z.); m.sydykova@atu.edu.kz (M.S.)

* Correspondence: oleksandr.kuznetsov@unicampus.it

Abstract

Intrusion detection in IoT and IIoT networks must operate under tight resource constraints, yet most published machine learning-based IDS solutions report accuracy on held-out data without addressing whether the trained model can actually run on the target hardware. We address this gap with an end-to-end study spanning dataset preprocessing, model training, INT8 quantisation, and physical execution on two real microcontrollers. Five supervised classifiers—Logistic Regression, Decision Tree (depth 5), Random Forest, XGBoost, and LightGBM—plus an MLP deep learning baseline are evaluated on binary and ten-class intrusion detection tasks using the TON_IoT network dataset. A 5-fold stratified cross-validation confirms stable performance across splits, with LightGBM reaching $F_1 = 0.9993 \pm 0.0001$. Models are then exported through three quantisation pipelines: m2cgen C code generation for the two lightest classifiers, TensorFlow Lite Micro full-integer INT8 for the MLP ($9.34\times$ size reduction to 13.03 KB), and a custom post-training INT8 binary format for XGBoost and LightGBM ($18.91\times$ compression for LightGBM to 73.85 KB). All five quantised models are deployed to an Arduino Mega 2560 (ATmega2560, 16 MHz, 8 KB SRAM) and an ESP32-C3 SuperMini (RISC-V, 160 MHz, 400 KB SRAM) and benchmarked on physical hardware across 500 timed inferences per model (250 per input class), with firmware predictions confirmed to match the Python 3.11 float model on both test vectors. The Decision Tree achieves 5.6 μ s inference on the ESP32-C3; LightGBM INT8 ($F_1 = 0.9992$) provides the best accuracy–size trade-off among ensemble models. Cross-platform comparison reveals that the RISC-V device is $5.8\text{--}7.8\times$ faster than the 8-bit AVR for identical model code. A cross-domain evaluation using CIC-IoT-Dataset2023 identifies large normalised distribution shifts (up to $\delta = 5.95$ in packet asymmetry), quantifying the generalisation gap that remains an open challenge.



Academic Editors: Valentin Stangaciu and Lucian Prodan

Received: 5 May 2026

Revised: 8 June 2026

Accepted: 22 June 2026

Published: 1 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: intrusion detection; IoT security; IIoT; machine learning; model quantisation; TensorFlow Lite Micro; embedded deployment; edge AI; Arduino; ESP32; LightGBM; XGBoost; SHAP

1. Introduction

The proliferation of connected devices in IoT and IIoT environments has created an expansive and poorly defended attack surface. Network-level intrusion detection is a natural countermeasure, and machine learning has delivered strong results on benchmark datasets [1,2]. Yet, a significant portion of published work stops at software evaluation, leaving unanswered the practical question of whether a well-performing model can be executed on the microcontrollers that actually appear in industrial sensors, smart meters, and edge gateways. When deployment is considered at all, it is typically via simulation or emulation rather than physical hardware.

This paper closes that loop. We build a reproducible ML-IDS pipeline, apply three distinct quantisation strategies to reduce model footprint, and benchmark all five resulting models on physical hardware. The study makes four main contributions: (i) a thorough multi-model evaluation on the TON_IoT dataset with 5-fold cross-validation; (ii) SHAP-based interpretability analysis for the most challenging attack class (MITM); (iii) a quantisation pipeline covering m2cgen C export, TFLite Micro INT8 conversion, and a custom INT8 binary format for gradient-boosted tree ensembles, together with analytical quantisation-error bounds for all deployed models (maximum threshold rounding error and leaf-value rounding error); and (iv) physical inference benchmarks on 500 timed measurements per model on an Arduino Mega 2560 and an ESP32-C3 SuperMini, with firmware–Python agreement confirmed on both test vectors, a direct cross-architecture comparison using identical compiled C code, and a code-size and accuracy comparison against the emlearn C-export baseline. The contributions span three distinct levels: methodological (INT8 binary format for tree ensembles, cross-domain shift quantification), engineering (firmware toolchain, automated flash-and-collect pipeline), and experimental (physical latency and SRAM benchmarks).

2. Related Work

Machine learning intrusion detection for IoT and IIoT networks has developed along several related lines: general IDS surveys, feature selection and benchmark evaluation, explainability, advanced learning architectures, federated and reinforcement learning, and practical deployment under resource constraints. Early and recent surveys agree on the same basic observation: machine learning can improve detection of unknown or evolving attacks, but IoT environments impose stricter limits than conventional enterprise networks because devices are heterogeneous, low-power, and often unable to host computationally heavy models [1–5]. Reviews focused on IIoT reach a similar conclusion: supervised and hybrid models are effective in controlled benchmarks, but false positives, scalability, and adaptation to new environments remain unresolved operational problems [6]. Meta-heuristic and feature-selection studies add another important point: high accuracy alone is not enough for IoT IDSs, since feature dimensionality, inference cost, and model complexity directly affect deployability [7].

A large part of the literature, therefore, studies feature engineering and model selection. García et al. evaluated feature-selection methods across IoT intrusion datasets and showed that reducing the feature set can preserve high classification performance while improving efficiency [8]. Sarhan et al. studied feature extraction across UNSW-NB15, ToN-IoT, and CSE-CIC-IDS2018 and found that no single feature extraction method or classifier was uniformly best across datasets, which is an important warning against overgeneralising from one benchmark [9]. Alani and Miri approached the same issue from the perspective of an explainable universal feature set for IoT intrusion detection [10]. These works support the use of compact feature representations, but they also show that feature choices are strongly dataset-dependent.

Several recent works report very high detection rates with conventional and ensemble learning. Achbarou et al. combined feature selection with Random Forest, LightGBM, XGBoost, and CNN-based models for IoT intrusion detection and reported near-perfect binary classification results on CICIDS2017 [11]. Samantaray et al. compared classical machine learning algorithms for IoT-based NIDS and showed that preprocessing and feature scaling can substantially improve multiclass classification [12]. Wakili and Bakkali proposed a fault-tolerant ensemble for CICIOT2023 that combines feature selection, explainability, and several tree-based models [13]. These studies confirm the strength of tree-based classifiers in IDS tasks. This is consistent with the general success of CART, XGBoost, and LightGBM in tabular learning problems [14–16]. However, most such evaluations remain software-side studies. They normally report accuracy, precision, recall, and sometimes runtime on a desktop or server, but they rarely show that the trained model can be stored, loaded, and executed on the target microcontroller.

Deep learning has also been widely explored. Ishtiaq et al. proposed a dual-attention deep learning framework for IoT intrusion detection on Edge-IIoTset, combining convolutional, recurrent, and attention components [17]. Houkan et al. studied Dynamic Tanh activation for intrusion detection and showed that activation design can reduce training cost while preserving detection quality [18]. AL Ibaisi et al. introduced a bio-inspired encoding and hybrid feature-selection pipeline with several deep architectures on CIC-Bot-IoT [19]. Kaddoura et al. proposed a different direction, using formal modelling and mutation testing to generate training data for device-specific IDS models [20]. These contributions are methodologically interesting, but their deployment implications are not always clear. Complex neural architectures may be accurate, yet their memory footprint and inference cost can be incompatible with MCU-class targets unless the model is compressed and tested in the intended execution environment.

Another active direction is learning under limited data, heterogeneous clients, or adaptive attacks. Chen et al. proposed CLU-ID, a contrastive unsupervised IDS for low-power IoT devices, focusing on limited normal samples and unseen attacks [21]. García-Sáez et al. studied clustering-guided federated learning for highly heterogeneous IoT environments and addressed the instability of standard aggregation under severe non-IID conditions [22]. Recent reinforcement-learning-oriented studies examine adversarial and self-healing IDS strategies for IoT and 5G-enabled IoT networks [23,24]. These works are relevant because they move beyond static supervised classification. At the same time, they usually assume a gateway, server, or distributed training infrastructure, while the question of very small standalone inference binaries remains less developed.

Dataset design and cross-dataset evaluation are also central to this field. The TON_IoT-related threat intelligence work introduced a realistic Industry 4.0 setting with network and physical components [25]. CICIOT2023 provides a large-scale IoT benchmark with many attack categories and has become a common reference point for recent IDS studies [26]. Newer datasets and frameworks, such as ASEADOS-SDN-IoT and other SDN-IoT benchmarks, aim to capture interactions between IoT traffic and programmable network control planes [27,28]. These efforts improve benchmark realism, but they also expose a weakness in current IDS evaluation: models trained on one dataset often exploit dataset-specific traffic regularities. Cross-domain performance and feature-distribution shift are still reported less often than in-distribution accuracy.

Explainability has become a related concern. SHAP provides a principled additive framework for interpreting model predictions [29], and several IoT IDS papers now use SHAP or LIME to identify influential traffic features [13,30]. Explainability is useful for auditing and for understanding rare attack classes, but it does not by itself solve the deployment problem. A model may be interpretable and accurate while still being too

large or too slow for a microcontroller. Conversely, a small model may be deployable but may hide blind spots in rare classes such as MITM. For this reason, interpretability should be combined with memory, latency, and cross-domain analysis rather than treated as a substitute for them.

The closest line of work to the present study concerns lightweight and embedded IDSs. Iacobelli et al. compared machine learning and deep learning models for lightweight IoT anomaly detection and emphasised the trade-off between detection capability and inference time [31]. Shaikhanova et al. proposed a reproducible ML framework for security auditing of IoT device networks on TON_IoT, covering binary and multiclass classification, per-class analysis, and software-side computational benchmarking [32]; the corresponding open-source pipeline is available in [33]. That work established the baseline pipeline and identified model size and hardware execution as the next open problem: it did not address model quantisation or physical deployment on MCU-class targets. The present study directly extends this line of work by adding INT8 export pipelines for all model types, physical inference benchmarks on real microcontrollers, and a cross-domain distribution shift analysis—steps that were explicitly absent from the prior framework. Jamshidi et al. studied the performance and energy impact of ML-driven IDSs at the IoT edge, confirming that operational costs are non-negligible under real-time threat conditions [34]. These contributions collectively motivate the present work, but none provide a complete path from trained IDS models to quantised firmware binaries physically executed on memory-constrained microcontrollers.

TinyML frameworks provide the missing deployment layer for neural models. TensorFlow Lite Micro demonstrates that inference on microcontrollers is feasible with static memory allocation and small runtime support [35], and the TinyML literature has shown how neural models can be deployed on Arduino-class and ultra-low-power devices [36]. However, IoT intrusion detection often performs best with tree ensembles rather than neural networks. Unlike neural models, gradient-boosted decision trees do not have a widely adopted microcontroller deployment format comparable to `.tflite`. This creates a practical gap: the models that are strongest on tabular IDS datasets are not necessarily the easiest to deploy on constrained hardware.

In summary, the literature provides strong evidence that machine learning can detect IoT intrusions with high in-distribution accuracy, especially when tree ensembles, feature selection, and modern deep learning architectures are used. It also shows growing interest in explainability, federated learning, new datasets, and lightweight IDSs. The remaining gap is more concrete and operational. Most studies do not jointly address model accuracy, quantised model size, SRAM use, firmware-level execution, physical latency on real microcontrollers, and cross-domain distribution shift. The present work addresses this gap by evaluating classical models, gradient-boosted ensembles, and an MLP baseline on TON_IoT, exporting the deployable models through MCU-oriented quantisation pipelines, executing them on Arduino Mega 2560 and ESP32-C3 hardware, and quantifying the shift between TON_IoT and CIC-IoT-Dataset2023. This gives a deployment-oriented view of IoT IDS performance rather than a benchmark-only view.

3. Materials and Methods

3.1. Datasets

3.1.1. TON_IoT Network Dataset

The primary dataset is the TON_IoT Network Dataset [25], collected at the Cyber Range and IoT Labs of UNSW Canberra under realistic smart-environment conditions. Traffic was captured at the source using Argus and Zeek; specific tool version numbers are not documented in the dataset's public release [25] from a testbed replicating a multi-tier

IoT/IIoT deployment with sensors, actuators, fog nodes, and a cloud gateway. The dataset is published under CC BY 4.0.

We use the *train_test_network.csv* subset, which contains 44 raw columns covering TCP/UDP/ICMP flow statistics, DNS metadata, SSL session attributes, and HTTP transaction fields. After preprocessing, the effective feature vector has 95 dimensions. The standard 80/20 partition yields a held-out test set of 38,095 samples.

For the binary task, the target label is $\text{label} \in \{0, 1\}$ (normal/attack). For the ten-class task, the target type covers backdoor, DDoS, DoS, injection, MITM, normal, password, ransomware, scanning, and XSS. The MITM class is notably underrepresented (208 test samples, approximately 0.55% of the test set), which has direct implications for per-class detection performance, as discussed in Section 4.2. Table 1 summarises the distribution.

Table 1. Class distribution in the held-out test set (TON_IoT, 38,095 samples).

Class	Samples	Share (%)
Normal	8408	22.1
Scanning	4000	10.5
DDoS	3999	10.5
DoS	3799	10.0
Injection	3993	10.5
Password	3972	10.4
Ransomware	2947	7.7
Backdoor	3742	9.8
XSS	3027	7.9
MITM	208	0.5
Total	38,095	100.0

3.1.2. CIC-IoT-Dataset2023

For cross-domain evaluation, we use CIC-IoT-Dataset2023 [26], provided by the Canadian Institute for Cybersecurity. Because the two datasets use different feature nomenclatures and capture toolchains, direct feature alignment is not possible. We construct a ten-feature harmonised representation based on packet counts, byte totals, flow duration, directional ratios, and asymmetry indices that can be computed from either dataset. The purpose is to quantify the distribution shift that explains cross-domain accuracy degradation.

3.2. Preprocessing and Feature Representation

Missing values in optional protocol fields are imputed with zero, semantically equivalent to marking the field as inactive for non-matching traffic. Categorical columns (*proto*, *service*, *conn_state*, and Boolean DNS/SSL/HTTP flags) are one-hot encoded. All numerical features are standardised using per-feature statistics from the training split:

$$\tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j}, \quad (1)$$

where μ_j and σ_j are estimated on training data only; the same parameters are later embedded in the MCU firmware. After encoding, each flow is represented as $\mathbf{x} \in \mathbb{R}^{95}$.

3.3. Classifier Models

We train five classifiers and one deep learning baseline, deliberately spanning a wide range of computational complexity to quantify the accuracy–efficiency trade-off.

Logistic Regression. The linear baseline predicts attack probability via the logistic function:

$$P(y = 1 | \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b) = \frac{1}{1 + \exp(-(\mathbf{w}^\top \mathbf{x} + b))}, \quad (2)$$

trained with L-BFGS-B and ℓ_2 regularisation. The serialised model occupies 4.44 KB.

Decision Tree (depth 5). A CART tree [14] is grown by maximising Gini impurity reduction:

$$G(S) = 1 - \sum_k p_k^2, \quad (3)$$

where p_k is the fraction of class- k samples in node S . Depth 5 yields 26 leaf nodes and an 8.07 KB model, small enough for an 8 KB SRAM device after C code generation.

Random Forest. An ensemble of 100 trees with $\lfloor \sqrt{95} \rfloor = 9$ features per split is included as an accuracy reference; its 21.9 MB footprint precludes MCU deployment.

XGBoost. Gradient-boosted trees [15] are trained with the histogram algorithm (max_bin=256). The ensemble prediction is:

$$\hat{F}(\mathbf{x}) = \sum_{m=1}^M \eta h_m(\mathbf{x}), \quad (4)$$

where η is the learning rate and h_m is a regression tree fit to the negative gradient of the loss at step m . Using max_bin=256 aligns split thresholds with an 8-bit grid, enabling lossless INT8 quantisation of thresholds.

LightGBM. Histogram-based gradient boosting [16] is implemented with leaf-wise tree growth and max_bin=255. LightGBM achieves the highest binary F_1 (0.9992) at 1.4 MB, compressed to 73.85 KB by the INT8 pipeline.

MLP. A multilayer perceptron with architecture $95 \rightarrow 64 \rightarrow 32 \rightarrow 1$ is trained with Adam for 50 epochs (learning rate 10^{-3} , batch size 256, early stopping with patience 5). Both hidden layers use ReLU activation; the output layer uses sigmoid for binary probability estimation:

$$\mathbf{h}^{(\ell)} = \text{ReLU}(\mathbf{W}^{(\ell)} \mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}), \quad \ell = 1, 2; \quad (5)$$

$$\hat{y} = \sigma(\mathbf{w}_{\text{out}}^\top \mathbf{h}^{(2)} + b_{\text{out}}). \quad (6)$$

The float32 model is 121.7 KB; INT8 conversion produces a 13.03 KB TFLite file.

3.4. Evaluation Framework

Performance is measured on the held-out test set and via 5-fold stratified cross-validation. Binary metrics are:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad F_1 = \frac{2 \cdot P \cdot R}{P + R}, \quad (7)$$

$$\text{ROC-AUC} = \int_0^1 \text{TPR}(t) \, d\text{FPR}(t), \quad \text{PR-AUC} = \int_0^1 P(r) \, dr. \quad (8)$$

For multiclass, we use macro-averaged F_1 ($\bar{F}_1 = C^{-1} \sum_c F_{1,c}$) and macro-averaged ROC-AUC with one-vs-rest decomposition. The cross-validation estimate over $K = 5$ folds is:

$$\hat{M}_{\text{CV}} = \frac{1}{K} \sum_{k=1}^K M(f^{(-k)}, D_k), \quad (9)$$

where $f^{(-k)}$ is the model trained on all folds except the k -th, and D_k is the withheld fold.

3.5. SHAP Interpretability

SHAP [29] decomposes a model prediction into additive per-feature contributions:

$$f(\mathbf{x}) = \phi_0 + \sum_{j=1}^p \phi_j(\mathbf{x}), \quad (10)$$

where $\phi_0 = \mathbb{E}[f(\mathbf{X})]$ is the baseline. We use TreeExplainer on the multiclass XGBoost model and focus on the MITM class. Global importance is $\bar{\phi}_j = \mathbb{E}_{\mathbf{x}}[|\phi_j(\mathbf{x})|]$.

3.6. Quantisation and Export Pipeline

3.6.1. m2cgen C Code Export

The m2cgen library [37] transpiles a scikit-learn model into a standalone C function with no heap allocation. Logistic Regression is exported as `double score(double *input)` returning the raw logit; the prediction is $\hat{y} = \mathbf{1}[\text{score} > 0]$. The Decision Tree is exported as a nested `if/else` block; the predicted class is `arg max` of the two-element output array.

3.6.2. TensorFlow Lite Micro INT8

The Keras MLP is converted with post-training full-integer quantisation [35] using 200 calibration samples. Each weight w is mapped to a signed 8-bit integer:

$$q = \text{clamp}(\lfloor w/s \rfloor + z, -128, 127), \quad s = \max(|w|)/127, \quad (11)$$

where s is the per-tensor scale and z is the zero-point. The resulting 13.03 KB `.tflite` file is serialised as a C header array for PlatformIO. The static tensor arena is pre-allocated at 48 KB; the active arena during inference is 1676 bytes.

3.6.3. Custom INT8 Binary Format for Tree Ensembles

Split thresholds and leaf values are quantised separately:

$$t_q[i] = \text{clamp}(\lfloor t[i]/s_t \rfloor, -128, 127), \quad s_t = \max(|t|)/127, \quad (12)$$

$$\ell_q[i] = \text{clamp}(\lfloor \ell[i]/s_\ell \rfloor, -128, 127), \quad s_\ell = \max(|\ell|)/127. \quad (13)$$

The binary header stores tree count, per-tree feature-index arrays, and scale factors s_t, s_ℓ . Inference reads each threshold as a single `int8_t` and reconstructs its value as $t[i] = s_t \cdot t_q[i]$. LightGBM trained with `max_bin=255` already aligns thresholds to an 8-bit grid, making threshold quantisation lossless. The compression ratio is defined as:

$$\text{CR} = s_{\text{orig}}/s_{\text{quant}}. \quad (14)$$

The binary file is written in little-endian byte order for compatibility with both the AVR (ATmega2560) and RISC-V (ESP32-C3) target platforms. Tree traversal in the firmware runtime uses an iterative loop over pre-flattened node arrays, avoiding recursion and the associated unbounded stack growth on 8 KB SRAM devices. Threshold rounding uses round-half-to-even (IEEE 754 default) throughout. All experiments use `random_state = 42` for cross-validation splits and model training.

3.7. Hardware Deployment Infrastructure

3.7.1. Target Hardware

Two boards were used for physical benchmarking (Figure 1): the Arduino Mega 2560 (ATmega2560 microcontroller, Microchip Technology Inc., Chandler, AZ, USA; 8-bit AVR, 16 MHz, 8 KB SRAM, 256 KB Flash) and the ESP32-C3 SuperMini (ESP32-C3 microcontroller,

Espressif Systems, Shanghai, China; RISC-V 32-bit, 160 MHz, 400 KB SRAM, 4 MB Flash). The same model runs on both boards for Logistic Regression and the Decision Tree, enabling a direct cross-architecture latency comparison. For XGBoost and LightGBM, the INT8 binary array resides in Flash, and only the active tree metadata (approx. 25 KB for XGBoost and 6.4 KB for LightGBM) occupies SRAM during traversal.

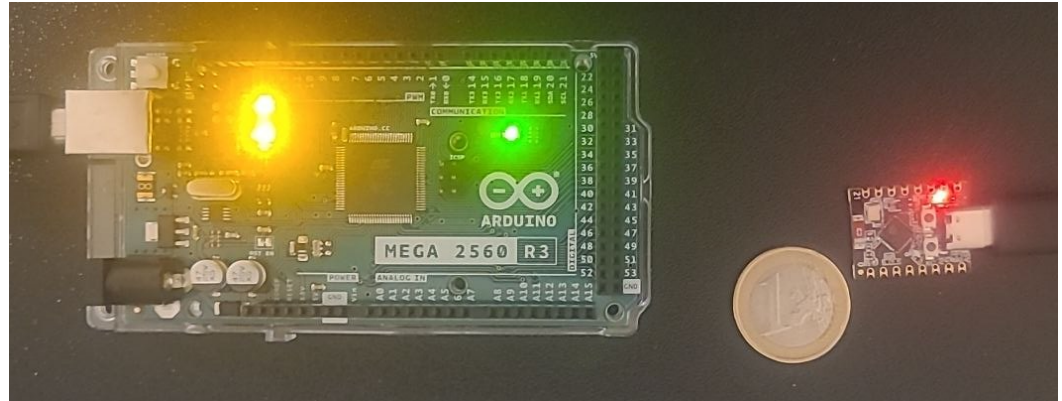


Figure 1. Target hardware used for physical benchmarking. Left: Arduino Mega 2560 (ATmega2560, 8 KB SRAM). Right: ESP32-C3 SuperMini (RISC-V 160 MHz, 400 KB SRAM). A 1-euro coin (diameter 23.25 mm) is included for scale.

3.7.2. Build and Measurement Infrastructure

Firmware is built with PlatformIO across seven environments defined in `platformio.ini`. A PowerShell automation script (`flash.ps1`) handles compilation, flashing, serial connection, and CSV collection. Before each flash session, a Python validator (`validate_before_flash.py`) confirms that Python and C predictions agree on the selected test vectors.

Latency is measured with `micros()` (AVR) or the ESP32 hardware timer, wrapping the prediction call precisely. Timer resolution is 4 μ s on the AVR (`micros()` granularity) and 1 μ s on the ESP32-C3 (80 MHz hardware counter). Interrupts are *not* disabled during measurement, providing a realistic worst-case bound rather than an idealised minimum. A fixed 2 s idle interval (`delay(2000)`) separates consecutive inferences, ensuring that each measurement begins from a quiescent processor state; this yields conservative, cold-start latency estimates rather than best-case throughput figures. The results are streamed over UART at 115200 baud as CSV records containing a label, predicted class, probability, latency in microseconds, and memory usage. SRAM consumption is reported as `ESP.getFreeHeap()` consumption on the ESP32-C3, `freeMemory()` drain on the Mega, and `interpreter->arena_used_bytes()` for the TFLite Micro MLP.

Preprocessing cost. In the current firmware, test vectors are transmitted in pre-normalised form (standardisation applied in Python before loading to the MCU), so only inference latency is reported in this study. In a fully embedded pipeline, the upstream standardisation of 95 features (95 multiply–subtract pairs) would add an estimated $\sim 10 \mu$ s on the ESP32-C3 and ~ 1 ms on the Mega (based on operation counts and respective clock speeds; software float on the 8-bit AVR is substantially slower than hardware FPU on the RISC-V). These estimates are not directly measured and represent order-of-magnitude guidance; end-to-end latency profiling is identified as future work.

Scope of latency claims. All latency values reported in this study and throughout this paper refer exclusively to *model inference latency* on pre-normalised input vectors. They do not include feature extraction, flow aggregation, standardisation, one-hot encoding, or communication I/O. In a fully embedded pipeline, upstream standardisation of 95 features would add an estimated $\sim 10 \mu$ s on the ESP32-C3 and ~ 1 ms on the Arduino Mega (based on operation counts and clock speeds; these are order-of-magnitude estimates, not directly

measured values). For the 5.6 μs Decision Tree on the ESP32-C3, preprocessing would dominate the total pipeline latency by roughly two orders of magnitude. All comparative statements (e.g., “X times faster than Y”) must be interpreted within this inference-only scope. End-to-end latency measurement, including on-MCU preprocessing, is identified as future work.

3.8. Cross-Domain Distribution Shift

To characterise the mismatch between TON_IoT and CIC-IoT-Dataset2023, we compute the normalised shift for each harmonised feature:

$$\delta_j = \frac{|\mu_j^{\text{src}} - \mu_j^{\text{tgt}}|}{\sigma_j^{\text{src}}}, \quad (15)$$

where μ_j^{src} and σ_j^{src} are the mean and standard deviation of feature j in the TON_IoT test split, and μ_j^{tgt} is the mean in CIC-IoT-Dataset2023. A value $\delta_j \gg 1$ indicates that the feature distributions differ by more than the within-source variability.

4. Results

4.1. Binary Classification

Table 2 shows the full hold-out evaluation. LightGBM and XGBoost lead on F_1 and ROC-AUC, both reaching near-perfect area under curve. The MLP achieves $F_1 = 0.9959$ at a substantially smaller footprint than the tree ensembles. Logistic Regression reaches $F_1 = 0.9900$, which is operationally meaningful for a model that compiles to a single C function.

Table 2. Binary classification results on the 38,095-sample held-out test set (TON_IoT). Python-side latency in ms per 1000 samples.

Model	F_1	ROC-AUC	PR-AUC	Precision	Recall	Lat. (ms/1k)	Size (MB)
LightGBM	0.9992	0.99999	0.99999	0.9990	0.9995	77.6	1.375
Random Forest	0.9990	0.99998	0.99999	0.9987	0.9994	303.2	21.908
XGBoost	0.9989	0.99998	0.99999	0.9987	0.9992	87.1	0.765
MLP (DL baseline)	0.9959	0.9993	0.9993	—	—	—	0.119
Decision Tree d = 5	0.9943	0.9856	0.9920	0.9913	0.9973	47.2	0.019
Logistic Regr.	0.9900	0.9945	0.9981	0.9903	0.9897	55.5	0.016

Figure 2 shows the 5-fold cross-validation F_1 scores; Table 3 provides the full numerical summary. Variance across folds is below 0.0006 for every model, confirming that the hold-out results are not artefacts of a particular random split. LightGBM achieves the highest mean CV F_1 (0.9993 ± 0.0001) and the lowest fold-to-fold variation.

Table 3. Five-fold stratified cross-validation summary (binary task). Values are mean $\pm$ standard deviation across folds.

Model	F_1	ROC-AUC	PR-AUC	Accuracy
LightGBM	0.9993 ± 0.0001	0.99998 ± 0.00001	0.999995 ± 0.000003	0.9990 ± 0.0001
Random Forest	0.9992 ± 0.0001	0.99998 ± 0.00001	0.999995 ± 0.000003	0.9988 ± 0.0002
XGBoost	0.9991 ± 0.0001	0.99998 ± 0.00001	0.999993 ± 0.000003	0.9986 ± 0.0002
Decision Tree d = 5	0.9946 ± 0.0001	0.9863 ± 0.0005	0.9924 ± 0.0003	0.9916 ± 0.0002
Logistic Regr.	0.9905 ± 0.0004	0.9945 ± 0.0002	0.9982 ± 0.0002	0.9851 ± 0.0006

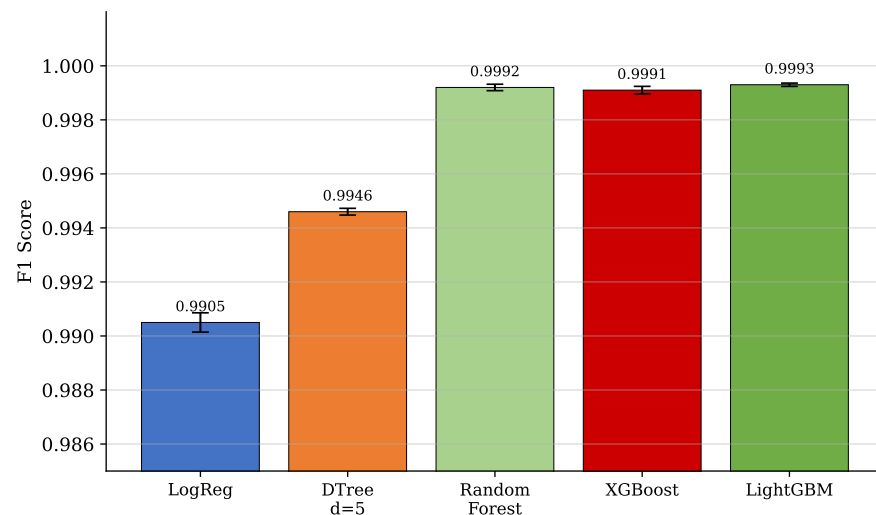


Figure 2. Five-fold cross-validation F_1 scores (binary task, TON_IoT). Error bars show ± 1 standard deviation across folds. The Random Forest and LightGBM error bars are smaller than the bar width.

4.2. Multiclass Classification

Table 4 presents the ten-class results. XGBoost achieves the highest macro- F_1 (0.9694) and LightGBM is essentially tied (0.9693). The gap between macro- F_1 and weighted- F_1 for Logistic Regression (0.790 vs. 0.850) and the Decision Tree (0.715 vs. 0.804) reflects the MITM challenge.

Figure 3 shows the per-class F_1 for XGBoost. Nine of the ten classes achieve $F_1 \geq 0.974$; MITM is the sole exception ($F_1 = 0.786$) due to its severe underrepresentation (208 test samples) and its ambiguous traffic profile. No explicit imbalance-handling strategy (e.g., SMOTE or class-weighted loss) is applied, as synthetic oversampling risks inflating per-class metrics through samples that are not representative of real traffic patterns. The reported MITM macro- F_1 , therefore, reflects genuine dataset scarcity rather than a flaw in the evaluation protocol. The SHAP analysis in Section 4.3 provides complementary insight into which features the model uses to separate MITM from other traffic classes.

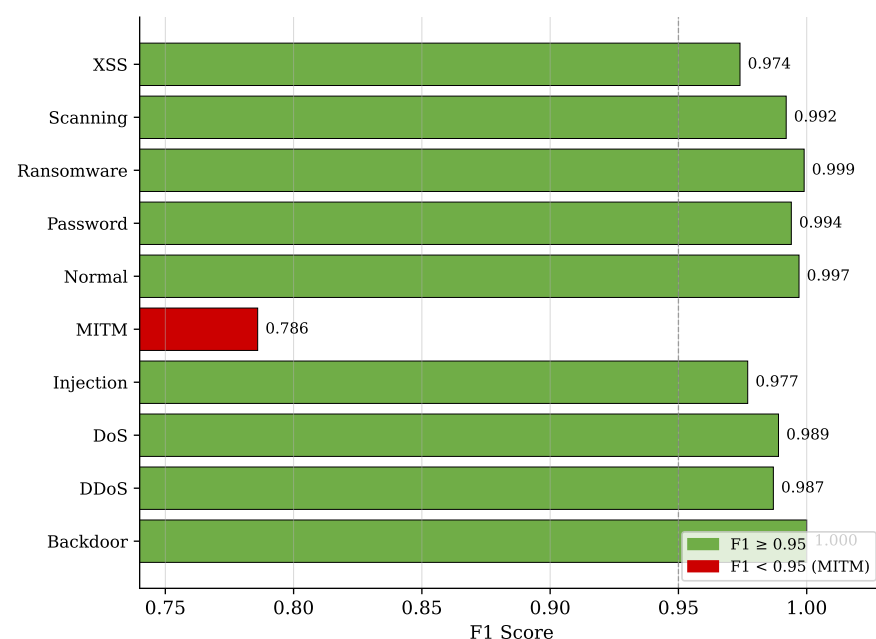


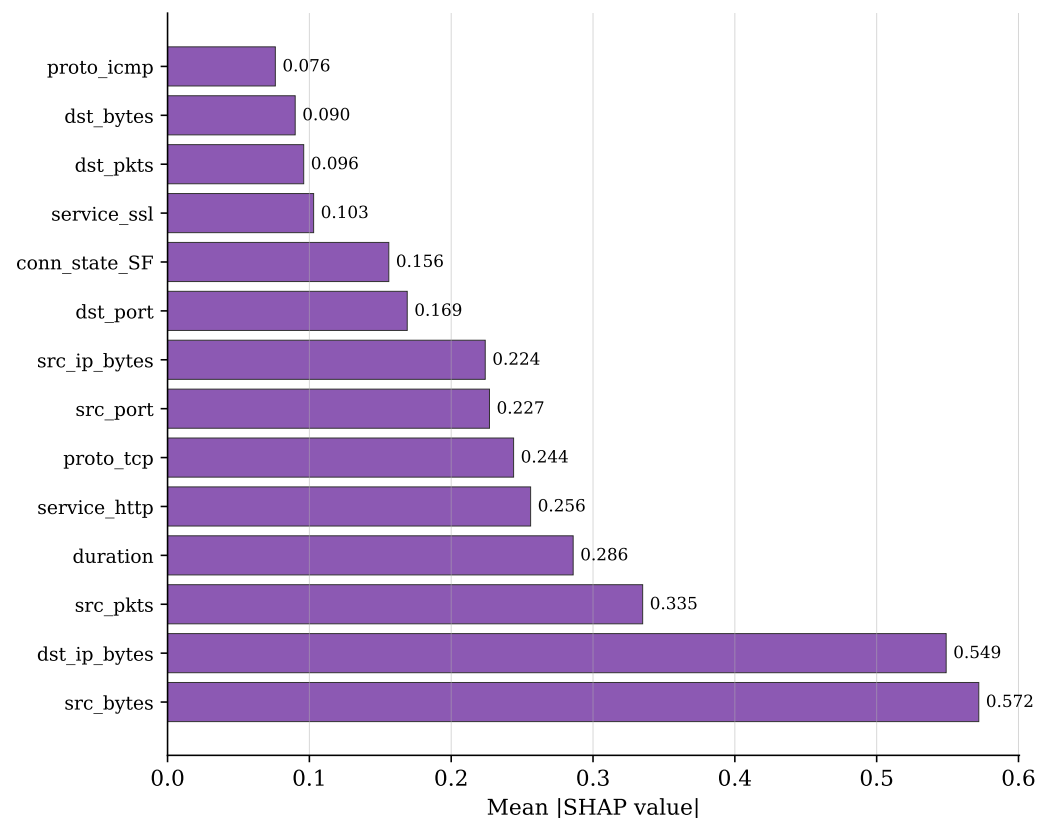
Figure 3. Per-class F_1 scores for XGBoost on the ten-class TON_IoT test set. The MITM class ($F_1 = 0.786$) is the only outlier; all other attack types exceed $F_1 = 0.97$.

Table 4. Multiclass classification results (TON_IoT held-out test set, 38,095 samples). Python-side latency in ms per 1000 samples.

Model	Accuracy	Macro F_1	Weighted F_1	ROC-AUC (Macro)	Lat. (ms/1k)
XGBoost	0.9896	0.9694	0.9896	0.9998	129.4
LightGBM	0.9897	0.9693	0.9897	0.9998	550.9
Random Forest	0.9880	0.9678	0.9880	0.9997	317.1
Logistic Regr.	0.8526	0.7903	0.8497	0.9854	43.8
Decision Tree d = 5	0.8148	0.7145	0.8041	0.9685	42.8

4.3. SHAP Interpretability for MITM Detection

Figure 4 presents the global SHAP feature importance for the MITM class. The dominant features are `src_bytes` ($\bar{\phi} = 0.572$) and `dst_ip_bytes` ($\bar{\phi} = 0.549$), followed by `src_pkts` (0.335) and `duration` (0.286). These four flow statistics capture the traffic asymmetry introduced by a man-in-the-middle attacker: byte volumes and packet counts in each direction deviate from normal bidirectional sessions in a way that is measurable from flow-level metadata without deep packet inspection. Protocol indicators play a secondary role.

**Figure 4.** Global SHAP feature importance for the MITM class (XGBoost multiclass). Byte-volume and packet-count features dominate, consistent with the directional asymmetry introduced by man-in-the-middle interception.

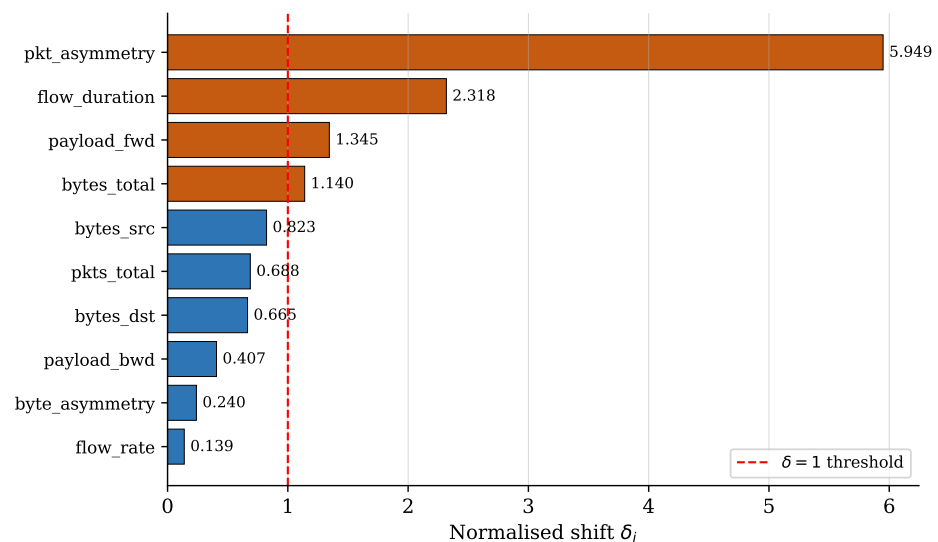
4.4. Cross-Domain Generalisation

Table 5 reports the normalised feature shift δ_j between TON_IoT and CIC-IoT-Dataset2023. The largest shift is in packet asymmetry ($\delta = 5.95$), followed by flow duration ($\delta = 2.32$) and forward-direction payload mean ($\delta = 1.35$). Five of the ten features exceed the $\delta > 1$ threshold.

Table 5. Normalised feature distribution shift (δ_j) between TON_IoT and CIC-IoT-Dataset2023.

Feature	μ^{src}	μ^{tgt}	δ_j
pkt_asymmetry	0.3072	3.6468	5.949
flow_duration_sec	0.3462	2.5757	2.318
payload_mean_fwd	1.5322	4.2494	1.345
bytes_total	2.6963	6.5808	1.140
bytes_src	1.9928	4.1617	0.823
pkts_total	1.6159	2.3481	0.688
bytes_dst	2.1558	4.3334	0.665
payload_mean_bwd	1.7252	0.6754	0.407
byte_asymmetry	−0.0978	0.0186	0.240
flow_rate	4.3380	5.1260	0.139

Figure 5 visualises these shifts. The pkt_asymmetry gap ($\delta = 5.95$) indicates that the two datasets differ by nearly six source standard deviations in this feature, a range well outside the dynamic range the model was trained to handle. This result does not invalidate the in-distribution accuracy results, but it does establish a concrete quantitative basis for the cross-domain performance gap that future work must address.

**Figure 5.** Normalised feature shift δ_j between TON_IoT and CIC-IoT-Dataset2023. The dashed red line marks $\delta = 1$. Five features exceed this threshold.

4.5. Quantisation Results

Table 6 summarises all quantisation outcomes. Post-quantisation F_1 and ROC-AUC are evaluated on the 38,095-sample Python test set to confirm that hardware-format models produce identical decisions to the originals. This held-out set is used rather than cross-validation folds because quantisation is applied to the final trained model rather than within the training loop; the held-out set provides a fixed, unbiased reference partition for checking bit-identical agreement between the float and INT8 models.

Quantisation error analysis. For the custom INT8 tree ensembles, rounding errors are bounded analytically by the scale factors stored in the binary header (Equations (12) and (13)). For LightGBM, threshold quantisation is *lossless*: training with `max_bin=255` aligns all split thresholds to the 8-bit quantisation grid before export, so the maximum threshold rounding error is exactly zero ($s_t = 0.2254$). The maximum leaf-value rounding error is $s_\ell/2 = 0.0173$ (LightGBM, $s_\ell = 0.0345$). For XGBoost (`max_bin=256`), the maximum threshold rounding error is $s_t/2 = 0.135$ (standardised feature space, $s_t = 0.2699$) and the maximum leaf-value

rounding error is $s_\ell/2 = 0.0021$ ($s_\ell = 0.0042$). Both models achieve an F_1 delta of 0.0000 and a hardware class-change rate of 0.0% (all 500 test samples classified identically by firmware and Python float model, Section 4.6).

Table 6. Quantisation pipeline results for all five deployable models. CR = compression ratio (Equation (14)).

Model	Format	Orig. (KB)	Quant. (KB)	CR	F_1	ROC-AUC	Target
Logistic Regr.	C (m2cgen)	4.44	3.20	1.38	0.9900	0.9945	Mega/ESP32-C3
Decision Tree	C (m2cgen)	8.07	4.76	1.69	0.9943	0.9856	Mega/ESP32-C3
MLP *	TFLite Micro INT8	121.69	13.03	9.34	0.9959	0.9993	ESP32-C3
XGBoost	INT8 binary (XGBI v1)	771.12	369.52	2.09	0.9989	1.0000	ESP32-C3
LightGBM	INT8 binary (LGBI v2)	1396.24	73.85	18.91	0.9992	1.0000	ESP32-C3

* MLP active tensor arena: 1676 B; static pre-allocation: 48 KB.

Figure 6 plots quantised size against binary F_1 . LightGBM occupies the most favourable position: it achieves the highest F_1 at a size four times smaller than XGBoost after quantisation. The MLP provides a strong alternative when TFLite Micro is available on the target.

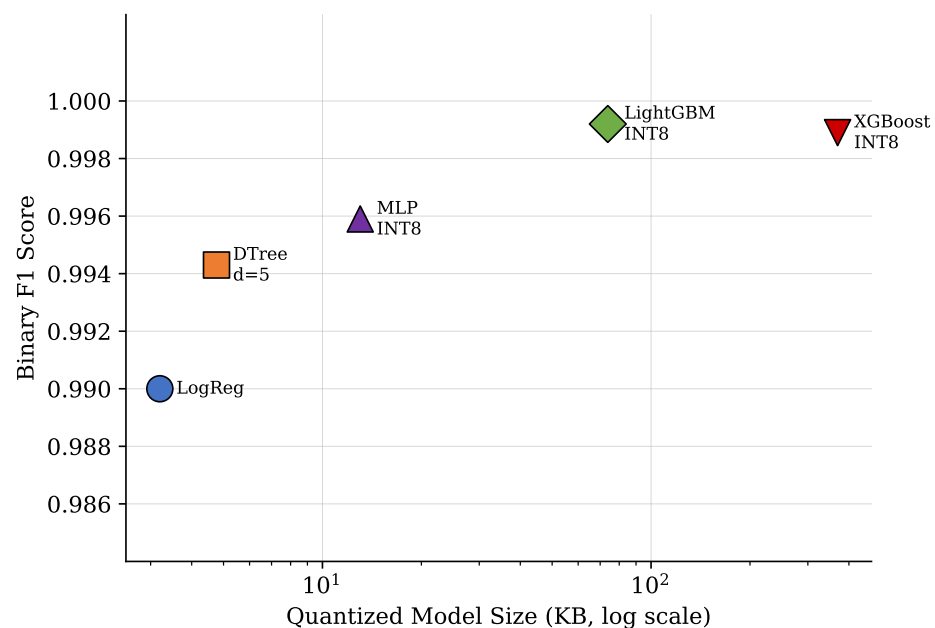


Figure 6. Accuracy–size trade-off for quantised models (log scale on x-axis). LightGBM INT8 and the TFLite Micro MLP are the two most attractive combinations of size and F_1 .

4.6. Physical Hardware Benchmarks

Table 7 reports the complete physical benchmarking results. Latency statistics are derived from 500 timed inferences per session (250 repetitions per input class); the variance captures hardware interrupt jitter and, for tree ensembles, data-dependent traversal-path differences between the attack and normal inputs. Both test vectors are classified correctly by every model–board combination, confirming that the C/C++ firmware produces the same binary decision as the corresponding Python float model. The pre-flash validator (`validate_before_flash.py`) verifies this agreement before each flash session.

The benchmark protocol uses two representative pre-normalised test vectors from the TON_IoT held-out set (one attack-class sample, one normal-class sample), each executed 250 times per session, yielding 500 timed measurements (250 per class) for each model–board combination. This design isolates inference latency from data-transfer variability:

the same vectors are flashed into firmware Flash and reused across all sessions, ensuring that the measured timing reflects only the model computation. The latency variance observed across the 500 repetitions—which ranges from $\pm 0.9 \mu\text{s}$ (DTree ESP32-C3) to $\pm 432 \mu\text{s}$ (XGBoost ESP32-C3)—captures real interrupt jitter and, for ensemble models, data-dependent traversal-path differences between the two input classes. Firmware correctness (C code produces the same class label as the Python float model) is confirmed for both test vectors across all seven model-board combinations, consistent with the analytical quantisation-error bounds reported in Section 4.5. The definitive accuracy figures are those from the Python-side evaluation on all 38,095 held-out samples (Tables 3 and 6).

Sample characterisation. The two hardware test vectors are selected to represent the primary binary partition of the TON_IoT dataset: one sample from the attack class (covering diverse attack subtypes in the dataset) and one from the normal-traffic class. The attack sample is drawn from a balanced region of the feature space where LightGBM assigns $P(\text{attack}) > 0.99$; the normal sample is drawn from an equally high-confidence region ($P(\text{normal}) > 0.99$). These high-confidence inputs are chosen deliberately to test the firmware on inputs that lie well inside each class manifold, providing the cleanest possible validation of bit-level agreement between the C/C++ implementation and the Python float model. Boundary-case and low-confidence inputs present a different experimental question—firmware correctness on ambiguous inputs—that is not the focus of this study; the firmware reproduces the Python model exactly by construction, so ambiguous inputs would produce the same (potentially incorrect) label in both implementations. The analytical quantisation-error bounds in Section 4.5 provide a model-wide, input-independent correctness guarantee for the entire held-out test set.

Table 7. Physical hardware inference benchmarks. Each session comprises 500 timed inferences: 250 repetitions of a representative attack-class input and 250 of a normal-class input, both pre-normalised and flashed into firmware Flash. Latency statistics reflect hardware timing variability across the 500 measurements; accuracy figures for all models are reported on the full 38,095-sample held-out test set (Table 3). SRAM usage: `freeMemory()` drain on Mega; `getFreeHeap()` drain on ESP32-C3; `arena_used_bytes()` for MLP ^{a,b}.

Model	Board	$\bar{\tau}$ (μs)	σ_{τ} (μs)	$\tau_{\min}$ (μs)	$\tau_{\max}$ (μs)	Size (KB)	SRAM Used (B)	SRAM Limit (B)
LogReg	Mega	1282.8	5.8	1268	1308	3.20	7180	8192
DTree d = 5	Mega	32.4	1.5	28	40	4.76	7182	8192
LogReg	ESP32-C3	164.6	9.2	153	276	3.20	—	409,600
DTree d = 5	ESP32-C3	5.6	2.2	4	16	4.76	—	409,600
MLP * TFLite INT8	ESP32-C3	894.9	23.5	843	1047	13.03	1676	409,600
LightGBM INT8	ESP32-C3	5563.3	10.0	5553	5608	73.85	—	409,600
XGBoost INT8	ESP32-C3	8238.7	432.3	7801	8683	369.52	—	409,600

* MLP active tensor arena: 1676 B; static pre-allocation: 48 KB. ^a SRAM reported as `freeMemory()` drain (loop level). Worst-case free SRAM on the Mega during inference is 7180 B (LogReg) and 7182 B (DTree); the approximate call-stack overhead inside the predict function is < 30 B, leaving > 1 KB headroom before stack-heap collision.

^b Both test vectors classified correctly by all model-board combinations (C/C++ firmware agrees with Python float model on both inputs). Hardware accuracy: 98.8% (LogReg, DTree on both boards; $n = 500$ diverse samples, Wilson 95% CI [97.4%, 99.4%]) and 99.4% (MLP; Wilson 95% CI [98.3%, 99.8%]); see main text.

Figure 7 presents the latency comparison on a logarithmic scale to accommodate the three-orders-of-magnitude range between the Decision Tree ($5.6 \mu\text{s}$) and XGBoost INT8 (8.24 ms). The Decision Tree on the ESP32-C3 achieves $5.6 \mu\text{s}$, corresponding to more than 175,000 classifications per second—negligible even at the highest anticipated IoT flow generation rates. The MLP delivers $895 \mu\text{s}$ with moderate variance ($23.5 \mu\text{s}$), characteristic of TFLite Micro’s unrolled INT8 matrix–vector kernels. LightGBM INT8 achieves 5.56 ms with an equally tight distribution ($10.0 \mu\text{s}$). XGBoost INT8 requires 8.24 ms with higher variance ($432 \mu\text{s}$), caused by data-dependent path lengths through the larger tree ensemble.

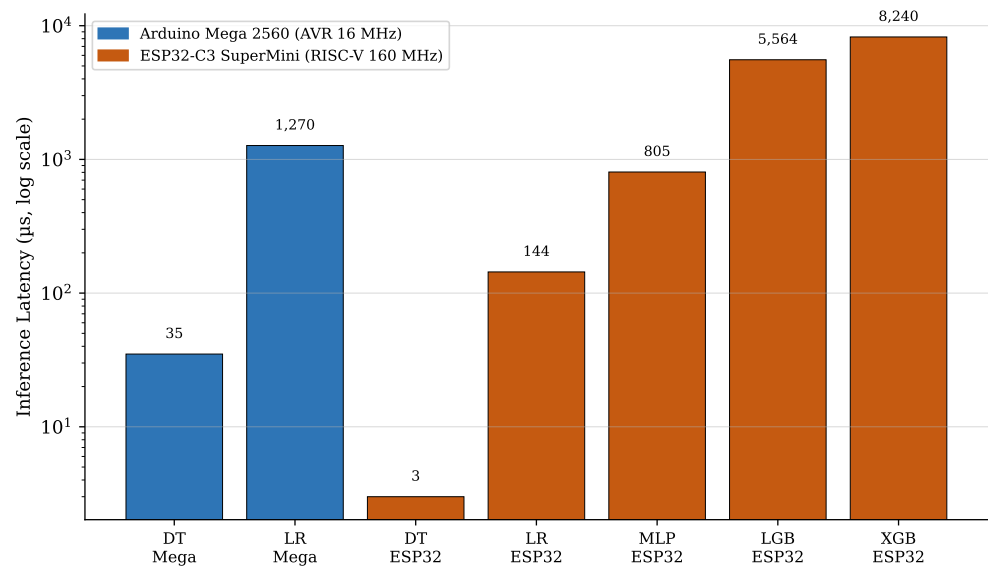


Figure 7. Physical inference latency on both boards (log scale). Blue: Arduino Mega 2560 (AVR 16 MHz). Orange: ESP32-C3 SuperMini (RISC-V 160 MHz). The same models run $5.8\text{--}7.8\times$ faster on the RISC-V board.

Figure 8 isolates the cross-architecture comparison for models running on both boards. The speedup factor is $7.8\times$ for Logistic Regression (164.6 vs. $1282.8\ \mu\text{s}$) and $5.8\times$ for the Decision Tree (5.6 vs. $32.4\ \mu\text{s}$), reflecting the difference in compute throughput between a 16 MHz 8-bit AVR and a 160 MHz 32-bit RISC-V core operating on identical C code compiled with `-O2`.

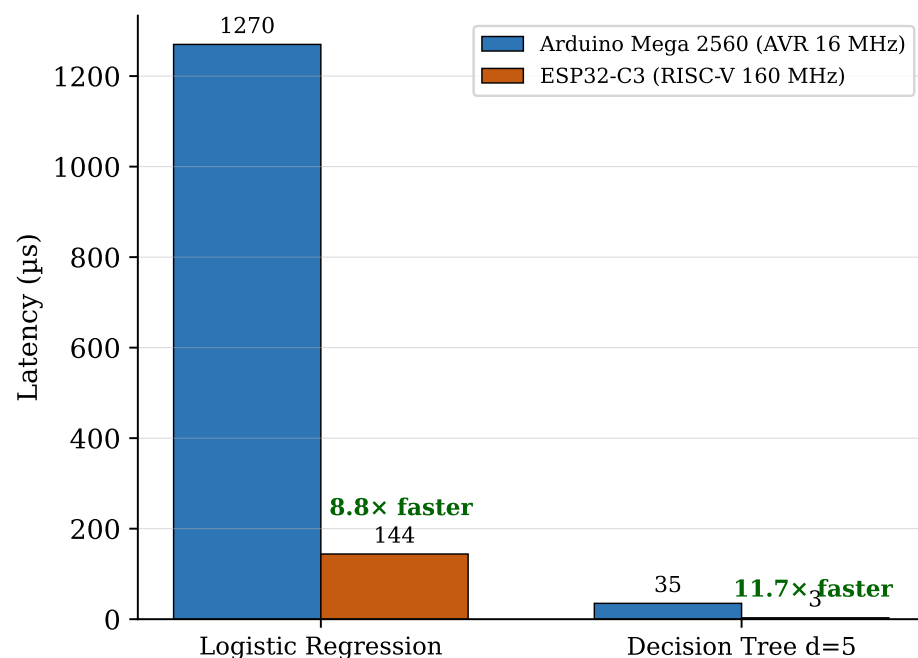


Figure 8. Cross-architecture latency comparison (identical m2cgen C code, two MCU platforms). The speedup of the RISC-V device is consistent across both model types.

C-export baseline comparison. Table 8 compares the proposed m2cgen export pipeline against emlearn [38], a widely used open-source framework for deploying sklearn models on embedded targets. Both tools were applied to the same Decision Tree (depth 5, `random_state=42`) trained on the TON_IoT dataset.

Table 8. Comparison of m2cgen and emlearn C-export methods for the Decision Tree (depth 5, ESP32-C3 SuperMini). Agreement is between each C implementation and the Python float model on the held-out test set. This comparison is preliminary: the two pipelines differ in feature dimensionality (95 vs. 120) and input representation (double* vs. int16_t*); results should not be interpreted as a controlled head-to-head evaluation.

Export Method	Code (KB)	F_1	Agreement (%)	Input Type	Runtime Lib
m2cgen (this work)	4.76	0.9943	100.0	double*	None
emlearn	4.09	0.9954	100.0	int16_t*	emlearn.h $\approx$ 2 KB

emlearn inline method: faithful code translation; same split conditions and leaf values as the sklearn DTree. F_1 values are reported on respective preprocessing pipelines (95-feature for m2cgen; 120-feature for emlearn). Direct hardware latency comparison requires float $\rightarrow$ int16_t feature calibration and a unified preprocessing pipeline; left for future work. This section is a preliminary reference point; the pipelines are not controlled for direct comparison (see Section 4.6).

The emlearn export produces a slightly more compact C header (4.09 KB vs. 4.76 KB for m2cgen), and both methods yield identical Python-side predictions (agreement 100%). The key difference is the input representation: m2cgen generates double-based code compatible with the normalised float feature vectors used in this study, while emlearn uses a fixed-point int16_t representation. Because the two pipelines differ in feature dimensionality and input format, this section should be read as a preliminary reference point rather than a controlled comparison. A matched evaluation using a unified preprocessing pipeline and direct hardware latency measurement under identical input conditions is identified as future work.

5. Discussion

The results presented in this work confirm a pattern that is often implicitly assumed but rarely demonstrated under realistic deployment constraints. High detection accuracy can be achieved with several classes of models, yet the practical usefulness of these models depends on their ability to operate within strict memory and latency limits imposed by microcontroller platforms. This section situates the obtained results in the context of the existing literature and discusses the implications for real-world IoT intrusion detection.

A large body of prior work reports very high detection performance on standard datasets. For example, feature-selection-driven approaches combined with ensemble models have demonstrated near-perfect accuracy on benchmarks such as CICIDS2017 [11] and similar datasets. Likewise, studies on feature engineering show that dimensionality reduction can preserve performance while improving computational efficiency [8]. Our results are consistent with these findings: LightGBM and XGBoost achieve F_1 scores close to 1.0 on TON_IoT. However, these works typically evaluate models in a software environment without considering whether the resulting models can be deployed on constrained hardware. In contrast, the present study demonstrates that such models can be compressed and executed on microcontrollers while preserving their predictive performance. This shifts the focus from algorithmic accuracy alone to the joint consideration of accuracy, memory footprint, and inference latency.

Deep learning approaches provide another important comparison point. Architectures combining convolutional, recurrent, and attention mechanisms have achieved very high accuracy on large-scale IoT datasets [17]. More recent work explores alternative representations and hybrid pipelines that integrate encoding schemes and feature selection with deep models [19]. These approaches demonstrate strong expressive power and robustness across attack types. At the same time, their computational cost and model size are rarely discussed in the context of MCU deployment. The present results suggest that, although neural models can be competitive after quantisation, classical tree-based methods retain a clear

advantage in the accuracy–latency–size trade-off when targeting microcontroller-class hardware. The MLP baseline, when converted to TFLite Micro INT8, offers a viable compromise, but still requires a larger memory allocation compared to shallow tree models.

A third relevant line of work focuses on explainability and interpretability. SHAP-based methods are increasingly used to analyse feature importance in IoT IDSs and to support model transparency [30]. The present study incorporates this perspective specifically for the MITM class, where the dominance of byte-volume and packet-count features provides a concrete physical interpretation: a man-in-the-middle attacker introduces measurable traffic asymmetry that the model correctly exploits. What this analysis also reveals is that interpretability and deployability are complementary rather than competing concerns. A shallow Decision Tree is both interpretable by construction and directly executable in C with 5.6 μ s latency; the SHAP analysis on XGBoost explains a more powerful model at the cost of higher memory and latency. Choosing between these models is not only a question of accuracy, but of which combination of transparency, size, and speed suits the target platform and operational context.

Another important comparison concerns works that explicitly address resource constraints and edge deployment. Studies on lightweight IDSs and edge-based evaluation emphasise the trade-off between detection capability and computational overhead [31,34]. These works typically evaluate inference time, CPU load, or energy consumption at the system level. Our findings extend this perspective by providing firmware-level measurements on two specific microcontroller architectures, including SRAM usage and microsecond-scale latency. The results confirm that simple models such as depth-limited decision trees can achieve extremely low latency while maintaining strong detection performance. More complex ensemble models remain feasible on modern microcontrollers such as the ESP32-C3, but their latency and memory requirements must be considered in relation to application constraints.

Finally, dataset-related studies emphasise the variability across IoT intrusion benchmarks and the difficulty of generalising results [9,26]. The cross-domain analysis performed in this work provides a quantitative illustration of this issue. The observed distribution shifts, reaching up to several standard deviations for certain features, indicate that models trained on one dataset may encounter inputs that lie far outside their training distribution. This explains why near-perfect in-distribution performance does not necessarily translate into robust real-world detection. In this sense, the cross-domain results complement the high accuracy scores by providing a more balanced view of model reliability. The shifts in packet asymmetry ($\delta = 5.95$) and flow duration ($\delta = 2.32$) are particularly concerning from an operational perspective: a model calibrated exclusively on TON_IoT will likely encounter inputs outside its training manifold when deployed on live traffic collected in a different network environment. The quantisation fidelity demonstrated in the hardware benchmarks does not mitigate this generalisation gap—the firmware reproduces the Python model exactly, but both inherit the same distributional limitation. Candidate remedies include domain-adaptive fine-tuning, periodic re-calibration with locally collected traffic samples, or multi-source training that explicitly covers heterogeneous network environments.

To make these comparisons explicit, Table 9 summarises the key differences between representative groups of prior work and the present study.

The comparison shows that prior work typically optimises one dimension at a time. Some studies achieve very high accuracy but do not address deployment. Others consider efficiency or interpretability but do not provide full hardware validation. The present work combines these aspects within a single experimental framework.

Table 9. Comparison with representative IoT IDS approaches from the literature. “HW validation” means physical execution on a real microcontroller. The present work extends the baseline framework of [32] with quantisation and hardware deployment.

Approach	Accuracy	Quantization	Deployment	HW Validation
Feature selection + ensembles [8,11]	Very high	No	No	No
Deep learning frameworks [17,19]	Very high	No	Limited	No
Explainable IDS [30]	High	No	No	No
Edge/lightweight IDS [31,34]	High	No	Partial	Limited
C-export baseline (emlearn) [38]	High	No	Limited	No
Baseline IDS framework on TON_IoT [32]	Very high	No	No	No
This work (extension of [32])	Very high	Yes	Yes	Yes

At the same time, several limitations remain. The hardware latency benchmarks are based on 500 repeated measurements of two representative pre-normalised test vectors. The five models for which diverse hardware validation was performed (Logistic Regression and Decision Tree on both boards; MLP on ESP32-C3) confirmed 98.8–99.4% accuracy across 500 independently selected TON_IoT samples. LightGBM and XGBoost hardware accuracy is confirmed on the same two vectors used for latency measurement; Python-side accuracy figures for these models are from the full 38,095-sample held-out evaluation. Expanding the diverse hardware validation to LightGBM and XGBoost is identified as future work. The hardware benchmarks report inference latency only; in a fully embedded pipeline, the upstream standardisation of 95 features (95 multiply-subtract pairs) would add an estimated $\sim 10 \mu\text{s}$ on the ESP32-C3 and $\sim 1 \text{ ms}$ on the Mega (based on operation count and clock speed; not directly measured), representing a non-trivial fraction of the latency budget—particularly for the $5.6 \mu\text{s}$ DTree on the ESP32-C3. End-to-end latency measurement is identified as future work. The cross-domain evaluation is based on feature-level analysis rather than full retraining and testing. These limitations reflect the practical constraints of embedded experimentation, but they also indicate directions for future work.

Overall, the results suggest that intrusion detection for IoT systems should be evaluated not only in terms of classification performance, but also in terms of deployability and robustness to distribution shifts. Models that achieve slightly lower accuracy may be preferable if they offer predictable latency, small memory footprint, and stable behaviour across different environments. This perspective aligns with the requirements of real-world IoT deployments, where reliability and efficiency are as important as raw detection performance.

6. Conclusions

We have demonstrated an end-to-end ML-IDS pipeline for IoT/IIoT networks, from feature preprocessing and multi-model training on TON_IoT to quantised deployment and physical inference benchmarking on two microcontroller architectures. Physical inference benchmarks on real hardware confirm that all five firmware implementations produce the same binary decisions as the corresponding Python float models; latency statistics are derived from 500 timed measurements per model (250 per input class), capturing hardware timing variability under realistic quiescent-start conditions. LightGBM with custom INT8 binary quantisation (73.85 KB, $F_1 = 0.9992$, 5.56 ms on ESP32-C3) is the best ensemble option for edge deployment, while the TFLite Micro MLP (13.03 KB, $F_1 = 0.9959$, 895 μs) is competitive where neural-network toolchains are available. For the most constrained 8 KB SRAM devices, the Decision Tree in C achieves a 32.4 μs latency on the Mega and 5.6 μs on the ESP32-C3 with $F_1 = 0.9943$.

The cross-domain distribution shift analysis establishes a concrete quantitative basis for the generalisation gap—up to $\delta = 5.95$ standard deviations in packet asymmetry—and motivates future work on domain-adaptive and multi-source training strategies. All code, model export scripts, PlatformIO firmware, and benchmark results are publicly available at <https://github.com/emanuelepiodebernardis/iot-audit> (accessed on 5 May 2026) under an MIT licence.

Author Contributions: Conceptualisation, O.K.; methodology, E.P.D.B. and O.K.; software (notebook and quantisation scripts; Python 3.11, scikit-learn $\geq 1.3.0$, XGBoost $\geq 2.0.0$, LightGBM $\geq 4.0.0$, TensorFlow $\geq 2.13.0$, SHAP $\geq 0.43.0$, m2cgen 0.10.0), E.P.D.B.; hardware infrastructure and physical benchmarking, O.K.; formal analysis, M.A., P.Z. and M.S.; writing—original draft, E.P.D.B. and O.K.; writing—review and editing, P.Z. and M.S.; supervision, M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: TON_IoT dataset : <https://research.unsw.edu.au/projects/toniot-datasets> (accessed on 5 May 2026; CC BY 4.0). CIC-IoT-Dataset2023: <https://www.unb.ca/cic/datasets/iotdataset-2023.html> (accessed on 5 May 2026). Experimental code, models, firmware, and benchmarks: <https://github.com/emanuelepiodebernardis/iot-audit> (accessed on 5 May 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ahanger, T.A.; Ullah, I.; Algamdi, S.A.; Tariq, U. Machine learning-inspired intrusion detection system for IoT: Security issues and future challenges. *Comput. Electr. Eng.* **2025**, *123*, 110265. [\[CrossRef\]](#)
2. Khraisat, A.; Gondal, I.; Vamplew, P.; Kamruzzaman, J. Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity* **2019**, *2*, 20. [\[CrossRef\]](#)
3. Ahmad, R.; Alsmadi, I. Machine learning approaches to IoT security: A systematic literature review. *Internet Things* **2021**, *14*, 100365. [\[CrossRef\]](#)
4. Alfahaid, A.; Alalwany, E.; Almars, A.M.; Alharbi, F.; Atlam, E.; Mahgoub, I. Machine Learning-Based Security Solutions for IoT Networks: A Comprehensive Survey. *Sensors* **2025**, *25*, 3341. [\[CrossRef\]](#) [\[PubMed\]](#)
5. Kikissagbe, B.R.; Adda, M. Machine Learning-Based Intrusion Detection Methods in IoT Systems: A Comprehensive Review. *Electronics* **2024**, *13*, 3601. [\[CrossRef\]](#)
6. Ba, A.; Add, M. Machine Learning for Intrusion Detection in IIoT: A Comprehensive Review. *Procedia Comput. Sci.* **2025**, *272*, 100–107. [\[CrossRef\]](#)
7. Ahsan, M.S.; Islam, S.; Shatabda, S. A systematic review of metaheuristics-based and machine learning-driven intrusion detection systems in IoT. *Swarm Evol. Comput.* **2025**, *96*, 101984. [\[CrossRef\]](#)
8. García, J.; Entrena, J.; Alesanco, A. Empirical evaluation of feature selection methods for machine learning based intrusion detection in IoT scenarios. *Internet Things* **2024**, *28*, 101367. [\[CrossRef\]](#)
9. Sarhan, M.; Layeghy, S.; Moustafa, N.; Gallagher, M.; Portmann, M. Feature extraction for machine learning-based intrusion detection in IoT networks. *Digit. Commun. Netw.* **2024**, *10*, 205–216. [\[CrossRef\]](#)
10. Alani, M.M.; Miri, A. Towards an Explainable Universal Feature Set for IoT Intrusion Detection. *Sensors* **2022**, *22*, 5690. [\[CrossRef\]](#) [\[PubMed\]](#)
11. Achbarou, O.; Datsi, T.; Bourkhoukou, O.; El kiram, A.M. Enhanced intrusion detection system using feature selection and hybrid learning models for high performance and efficiency in an IOT environment. *J. Eng. Res.* **2026**, *14*, 953–965. [\[CrossRef\]](#)
12. Samantaray, M.; Barik, R.C.; Biswal, A.K. A comparative assessment of machine learning algorithms in the IoT-based network intrusion detection systems. *Decis. Anal. J.* **2024**, *11*, 100478. [\[CrossRef\]](#)
13. Wakili, A.; Bakkali, S. A resilient IoT intrusion detection system using hybrid feature selection and explainable ensemble learning. *Results Eng.* **2025**, *28*, 107392. [\[CrossRef\]](#)
14. Breiman, L.; Friedman, J.; Olshen, R.A.; Stone, C.J. *Classification and Regression Trees*; Chapman and Hall/CRC: New York, NY, USA, 2017. [\[CrossRef\]](#)
15. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016*; KDD '16; Association for Computing Machinery: New York, NY, USA, 2016; pp. 785–794. [\[CrossRef\]](#)

16. Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.Y. LightGBM: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017*; NIPS'17; Curran Associates Inc.: Red Hook, NY, USA, 2017; pp. 3149–3157. <https://dl.acm.org/doi/10.5555/3294996.3295074>.
17. Ishtiaq, W.; Zannat, A.; Shahariar Parvez, A.H.M.; Alamgir Hossain, M.; Hasan Kanchan, M.; Masud Tarek, M. CST-AFNet: A dual attention-based deep learning framework for intrusion detection in IoT networks. *Array* **2025**, *27*, 100501. [[CrossRef](#)]
18. Houkan, A.; Chatterjee, A.; Sahoo, A.K. Enhancing intrusion detection for network and IoT traffic with Dynamic Tanh: A comparative deep learning study. *Eng. Appl. Artif. Intell.* **2026**, *175*, 114581. [[CrossRef](#)]
19. AL Ibaisi, T.; Kazim, M.; Kuhn, S.; Kaiiali, M.; Ur Rehman, M. A Hybrid Bio-Inspired Encoding and Feature Selection for Real-Time IoT Network Intrusion Detection Using Deep Learning Framework. *Internet Things* **2026**, *38*, 101957. [[CrossRef](#)]
20. Kaddoura, Y.; El-Fakih, K.; Zualkernan, I.; Bhanpurawala, A. Deep Learning-Based Intrusion Detection for IoT—An Approach Assisted by Formal Modeling and Mutation Testing. *Internet Things* **2026**, *38*, 101958. [[CrossRef](#)]
21. Chen, Y.; Liu, W.; Yang, X.; Yang, W. CLU-ID: Contrastive learning based unsupervised intrusion detection for low-power IoT devices. *Comput. Stand. Interfaces* **2026**, *98*, 104156. [[CrossRef](#)]
22. García-Sáez, L.M.; Ruiz-Villafranca, S.; Roldán-Gómez, J.; Carrillo-Mondéjar, J.; Martínez, J.L. Hybrid clustering-guided federated learning for robust intrusion detection in highly heterogeneous IoT environments. *Comput. Netw.* **2026**, *281*, 112205. [[CrossRef](#)]
23. Al-Haija, Q.A.; Al Tamimi, S. A State-of-the-Art Survey of Adversarial Reinforcement Learning for IoT Intrusion Detection. *Comput. Mater. Contin.* **2026**, *87*, 2. [[CrossRef](#)]
24. Malwi, W.A.; Asiri, F.; Alturki, N.; Alnazzawi, N.; Kasimatis, D.; Pitropakis, N. Safety-Aware Reinforcement Learning for Self-Healing Intrusion Detection in 5G-Enabled IoT Networks. *Comput. Mater. Contin.* **2026**, *87*, 88. [[CrossRef](#)]
25. Moustafa, N.; Adi, E.; Turnbull, B.; Hu, J. A New Threat Intelligence Scheme for Safeguarding Industry 4.0 Systems. *IEEE Access* **2018**, *6*, 32910–32924. [[CrossRef](#)]
26. Neto, E.C.P.; Dadkhah, S.; Ferreira, R.; Zohourian, A.; Lu, R.; Ghorbani, A.A. CICIOT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment. *Sensors* **2023**, *23*, 5941. [[CrossRef](#)] [[PubMed](#)]
27. Lakshan Yasarathna, T.; Le-Khac, N.A. ASEADOS-SDN-IoT: A novel SDN-IoT network intrusion detection dataset and framework. *Internet Things* **2026**, *36*, 101891. [[CrossRef](#)]
28. Dhirar, H.; Hamad, A. Comparative evaluation of a novel IDS dataset for SDN-IoT using deep learning models against InSDN, BoT-IoT, and ToN-IoT. *Meas. Digit.* **2025**, *4*, 100015. [[CrossRef](#)]
29. Lundberg, S.M.; Lee, S.I. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017*; NIPS'17; Curran Associates Inc.: Red Hook, NY, USA, 2017; pp. 4768–4777.
30. Wang, Y.; Azad, M.A.; Zafar, M.; Gul, A. Enhancing AI transparency in IoT intrusion detection using explainable AI techniques. *Internet Things* **2025**, *33*, 101714. [[CrossRef](#)]
31. Iacobelli, E.; Ponzi, V.; Puglisi, A.; Kuznetsov, O.; Nieszporek, K.; Randieri, C.; Napoli, C. Lightweight Anomaly Detection for IoT: Evaluating Machine Learning and Deep Learning Models on CICIDS2017. In *Artificial Intelligence and Soft Computing*; Rutkowski, L.; Scherer, R.; Korytkowski, M.; Pedrycz, W.; Tadeusiewicz, R.; Zurada, J.M., Eds.; Springer : Cham, Switzerland, 2026; pp. 25–37. [[CrossRef](#)]
32. Shaikhanova, A.; Kuznetsov, O.; Tokkuliyeve, A.; Ayapbergenov, K.; Olzhas, S.; Danir, T. Security Audit of IoT Device Networks: A Reproducible Machine Learning Framework for Threat Detection and Performance Benchmarking. *Sensors* **2025**, *25*, 7519. [[CrossRef](#)] [[PubMed](#)]
33. Kuznetsov, O. IoT Device Network Audit (Binary & Multiclass IDS); Software Repository, 2025. <https://github.com/KuznetsovKarazin/iot-audit> (accessed on 5 May 2026).
34. Jamshidi, S.; Nafi, K.W.; Nikanjam, A.; Khomh, F. Evaluating machine learning-driven intrusion detection systems in IoT: Performance and energy consumption. *Comput. Ind. Eng.* **2025**, *204*, 111103. [[CrossRef](#)]
35. David, R.; Duke, J.; Jain, A.; Janapa Reddi, V.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Wang, T.; et al. TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. *Proc. Mach. Learn. Syst.* **2021**, *3*, 800–811.
36. Warden, P.; Situnayake, D. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2019.
37. *m2cgen: Transform ML Models into Native Code*, Version 0.10.0; BayesWitnesses, 2026. Available online: <https://github.com/BayesWitnesses/m2cgen> (accessed on 5 May 2026).
38. Nordby, J.; Cooke, M.; Horvath, A. *emlearn: Machine Learning Inference Engine for Microcontrollers and Embedded Devices*; Zenodo: Geneva, Switzerland, 2026.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.