iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: https://doi.org/10.22214/ijraset.2026.85421

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

Lightweight Intrusion Detection for Resource-Constrained IOT Devices Using Machine Learning

Rakhi Kumari, Priti Kumari, Khushbu Kumari, Mahi Kumari
Student, Department of Electronics Engineering, Government Polytechnic Sitamarhi, India

Abstract: ***Billions of low-cost Internet of Things (IoT) devices run on microcontrollers with kilobytes of memory and no room for conventional intrusion detection, which has made them easy targets for botnets such as Mirai. This paper presents the design of a lightweight, machine-learning-based intrusion detection system (IDS) that runs on the IoT device itself. A hook in the device's TCP/IP stack updates a few counters for every packet in constant time; every 2 s these are turned into 20 traffic and host features, using 64-bit bitmaps to estimate distinct addresses and ports. A depth-limited decision tree or an 8-bit quantized multilayer perceptron classifies each window as benign or malicious, and a 3-of-5 vote across consecutive windows decides whether to block the source and raise an MQTT alert. An optional gateway tier names the attack type. A review of 16 related studies shows that most machine-learning IDS work is evaluated offline on datasets, uses features that a microcontroller cannot compute, and rarely reports how models generalise across datasets. Design analysis shows that the multilayer perceptron needs 490 parameters and 464 multiply–accumulate operations per window, a full depth-8 tree at most 2.6 KB, and the feature state under 0.3 KB, all within the budget of a Class 1 constrained device; the voting rule reduces expected false alarms from 43 to under 0.001 per day at a 0.1% per-window false-positive rate, assuming independent windows. A validation protocol using TON_IoT, CICIoT2023, Edge-IIoTset and N-BaIoT, including cross-dataset tests and on-device measurements, is provided.***

IndexTerms - Intrusion detection, Internet of Things, constrained devices, TinyML, decision tree, quantization, botnet, ESP32-C3

I. INTRODUCTION

IoT devices such as cameras, smart plugs, sensors and controllers are usually built on low-cost microcontrollers, shipped with weak default settings and rarely updated. The Mirai botnet showed the consequence: it compromised large numbers of such devices by trying default passwords and used them in some of the largest distributed denial-of-service (DDoS) attacks recorded [1], [2]. Because IoT devices are now deployed in homes, hospitals, farms and factories, a compromised device is both a victim and a launch point for further attacks.

An intrusion detection system (IDS) monitors network or host activity and raises an alert when it observes malicious behaviour. Machine-learning IDSs learn the difference between benign and attack traffic from data rather than from hand-written signatures [4]. Most of them, however, assume a gateway or server with ample memory and processing power. The devices most in need of protection fall into the constrained-node classes defined in RFC 7228: Class 0 devices have well under 10 KiB of RAM and 100 KiB of flash, Class 1 about 10 KiB and 100 KiB, and Class 2 about 50 KiB and 250 KiB [3]. A model of several hundred kilobytes, or a feature set computed from complete flow records, cannot run on such a device.

Placing detection on the device itself has clear benefits. It sees encrypted and local traffic that a network sensor may miss, it can respond in milliseconds by blocking the offending source, and it keeps working when the device is cut off from the gateway. This paper designs an IDS for that setting. Its contributions are:

1) An on-device IDS architecture in which a packet hook updates fixed-size state in constant time per packet, and features are produced once per 2 s window.
2) A set of 20 traffic and host features chosen to be computable on a microcontroller, including bitmap-based estimates of distinct addresses and ports.
3) Two interchangeable tier-1 models — a depth-limited decision tree and an INT8 multilayer perceptron — with a k-of-n voting rule that controls false alarms, and an optional gateway tier for attack-type classification.
4) A review of 16 related studies, an analysis of memory, computation and false-alarm rate against RFC 7228 device classes, and a validation protocol including cross-dataset and on-device tests.

Section II reviews related work. Sections III to V describe the architecture, features and models. Section VI sets out the experimental method, Section VII presents the results and discussion, and Section VIII concludes.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

II. LITERATURE REVIEW

The 16 reviewed studies are grouped into the threat and surveys that motivate IoT intrusion detection, IDS designs for IoT networks, public datasets, and machine-learning IDSs deployed on constrained hardware. Table 1 summarises the studies that propose or evaluate a detection method.

A. Threats and surveys

Kolias et al. analysed the Mirai source code and related botnets and described how they scan for devices with default credentials and launch floods [1]. Antonakakis et al. measured Mirai's growth in detail and showed that it was built from simple, poorly secured devices such as cameras and routers [2]. Da Costa et al. surveyed more than 95 machine-learning IDS studies for IoT and compared their techniques and architectures [4].

B. Intrusion detection designs for IoT networks

Raza et al. proposed SVELTE for 6LoWPAN networks using the RPL routing protocol, with heavy analysis at the border router and lightweight modules on the nodes. The node modules added only about 1.76 KB of ROM and 0.365 KB of RAM, and detected sinkhole and selective-forwarding attacks in simulation [5]. Meidan et al. (N-BaIoT) trained a deep autoencoder per device type on benign traffic snapshots and detected Mirai and BASHLITE attacks from nine infected commercial devices as soon as they were launched [6]. Mirsky et al. (Kitsune) used an ensemble of small autoencoders over 115 incremental traffic statistics, learning without labels; on one core of a Raspberry Pi 3B it processed about 5,400 packets per second [7]. Doshi et al. showed that flow-based features reflecting IoT behaviour, such as packet size, inter-packet interval and the number of distinct destinations, let simple classifiers including decision trees detect DDoS traffic from consumer IoT devices with about 0.999 accuracy [8].

C. IoT intrusion datasets

Koroniotis et al. built the Bot-IoT dataset of botnet traffic for network forensics [9]. Alsaedi et al. released TON_IoT, which combines IoT and industrial IoT telemetry with operating-system logs and network traffic, with binary and attack-type labels [10]. Ferrag et al. released Edge-IIoTset, covering 14 attacks in five categories from more than ten types of IoT devices, for centralised and federated learning [11]. Neto et al. released CICIoT2023, in which 33 attacks in seven categories were launched by malicious IoT devices against others in a 105-device topology [12].

D. Machine-learning IDS on constrained devices

Alwaisi et al. trained naive Bayes, decision-tree, random-forest and k-nearest-neighbour models on a Raspberry Pi 4 and an Arduino Nano 33 BLE to detect flooding and resource-exhaustion attacks, with accuracy of about 96–99.6%; random-forest training was slow on the Arduino, and the board failed with more than about 4,000 training samples [13]. Javed et al. built a two-layer IDS for an ESP32 smart thermostat, with XGBoost on the device detecting denial-of-service and man-in-the-middle attacks with about 97.6% accuracy in 3.51 ms, while a decision tree took only about 11 µs but was less accurate [14]. Fusco et al. presented TinyIDS, which trains on board so that sensitive data stays on the device, and evaluated it on microcontroller-based devices with TON_IoT [15].
Lundqvist et al. compared a decision tree, LightGBM, a feed-forward network and a TinyML-optimised network on ToN_IoT; the TinyML network gave the best balance, with an F1 score of 0.976 and a model of 31 KB [16]. De Bernardis et al. quantised six models to INT8 and ran them on an Arduino Mega 2560 (8 KB SRAM) and an ESP32-C3. LightGBM reached an F1 score of 0.9992 at 73.85 KB, the INT8 MLP needed 13.03 KB, and the decision tree ran in about 5.6 µs on the ESP32-C3. A cross-dataset test on CICIoT2023, however, revealed a large distribution shift [17].

Table 1 Summary of reviewed IDS studies

Ref.	Year	Platform	Data	Method	Key result or limitation
[5]	2013	Contiki nodes + border router	Simulation (Cooja)	Rule-based, RPL graph	1.76 KB ROM, 0.365 KB RAM per node
[6]	2018	Network monitor	9 infected devices	Deep autoencoders	Detects Mirai, BASHLITE at launch
[7]	2018	Raspberry Pi 3B	Own captures	Autoencoder ensemble	~5,400 packets/s; unsupervised
[8]	2018	Home gateway	Own consumer IoT	DT, RF, KNN, SVM, NN	~0.999 accuracy for DDoS
[13]	2024	RPi 4, Nano 33 BLE	Own lab data	NB, DT, RF, KNN	96–99.6%; Arduino limit ~4k samples

©IJRASET: All Rights are Reserved

Ref.	Year	Platform	Data	Method	Key result or limitation
[14]	2024	ESP32 thermostat + cloud	TON_IoT + own	XGBoost (device)	~97.6%, 3.51 ms; DT 11 µs
[15]	2024	MCU devices	TON_IoT	On-board training	Data stays on the device
[16]	2025	Constrained IoT	ToN_IoT	DT, LightGBM, FNN	TinyML FNN: F1 0.976, 31 KB
[17]	2026	Mega 2560, ESP32-C3	TON_IoT, CICIoT2023	6 models, INT8	LightGBM F1 0.9992 (73.85 KB); shift across datasets

E. Research gap

Three gaps emerge. First, most models are trained on feature sets defined by the dataset authors, often computed from complete flows by tools such as packet-capture software on a server [10]–[12]; the same features cannot simply be computed on a microcontroller, so offline accuracy does not transfer to the device. Second, even compact models of 13–74 KB [16], [17] consume most of the flash budget of a Class 1 device [3], leaving little for the application. Third, generalisation across datasets is rarely tested, and when it is, accuracy drops sharply [17]. The proposed design addresses these by defining the features from what the device can observe, keeping the model within a few kilobytes, and making cross-dataset and on-device evaluation part of the method (Table 2).

Table 2 Gaps in existing work and how the proposed design addresses them

Gap	Evidence in reviewed work	Approach in this paper
Features not computable on the device	Dataset features from full flows [10]–[12]	20 features defined by the on-device code; datasets replayed through the same code
Model too large for Class 0/1 devices	13–74 KB models [16], [17]	Tree ≤ 2.6 KB or INT8 MLP ≈ 0.6 KB
Poor cross-dataset generalisation	Large shift TON_IoT → CICIoT2023 [17]	Train on several datasets plus device captures; report cross-dataset results
Alarm rate not controlled	Per-sample accuracy reported	k-of-n voting with an explicit false-alarm budget
Detection without response	Most studies stop at classification	Local blocking or rate-limiting plus MQTT alert

III. PROPOSED SYSTEM ARCHITECTURE

The IDS is a small module compiled into the device firmware alongside the normal application (Fig. 1). A hook in the lwIP TCP/IP stack passes the header fields of every received and transmitted packet to the IDS, which updates its window state and returns immediately. Every 2 s the module computes a feature vector, runs the tier-1 model and applies the voting rule. If an attack is confirmed, it blocks or rate-limits the offending source in the stack, closes services that should not be exposed, and publishes an MQTT alert [25]. A heartbeat every 60 s lets the gateway notice a device that has gone silent, which may itself indicate compromise.

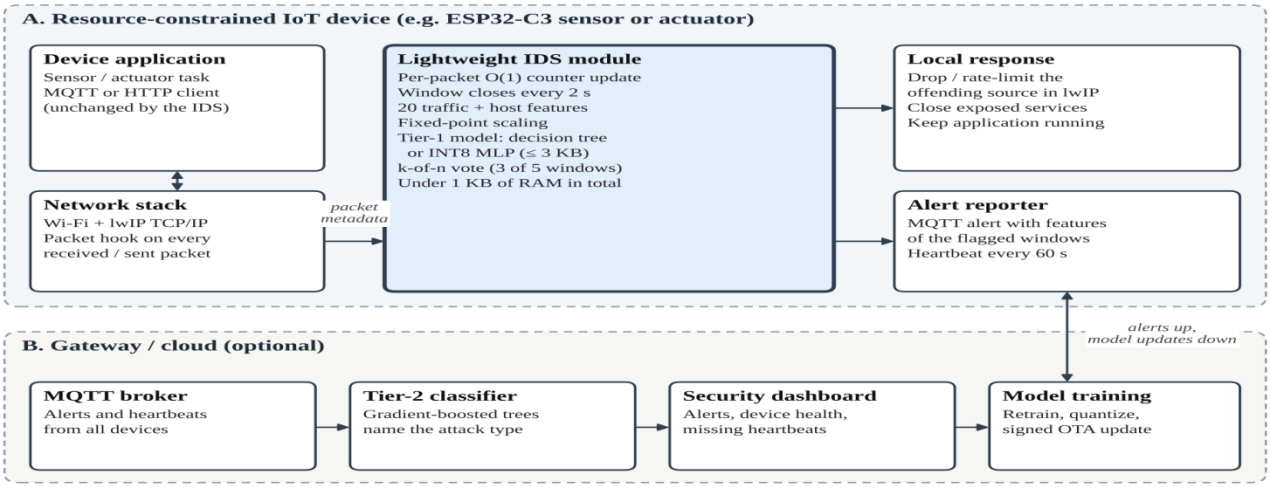


Fig. 1 Architecture of the on-device lightweight IDS with optional gateway tier

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

The gateway tier is optional. Where a gateway exists, it collects alerts from many devices, runs a larger gradient-boosted tree model [19], [20] on the reported feature vectors to name the attack type, and manages model updates, which are signed and installed over the air. Without a gateway, each device still detects and responds on its own.

IV. FEATURE DESIGN

Features are chosen so that each can be updated in constant time per packet and stored in a few bytes. Table 3 lists the 20 features. They capture the behaviours that distinguish common IoT attacks: floods raise packet rates and SYN or UDP counts, scans raise the number of distinct destination ports, brute-force logins raise failed connection attempts, and a compromised device that joins a botnet suddenly contacts many new destinations [8]. Two host features, the drop in free heap and the CPU load of the network task, capture resource-exhaustion attacks of the kind studied in [13]. Fig. 2 shows how the features are produced and used.

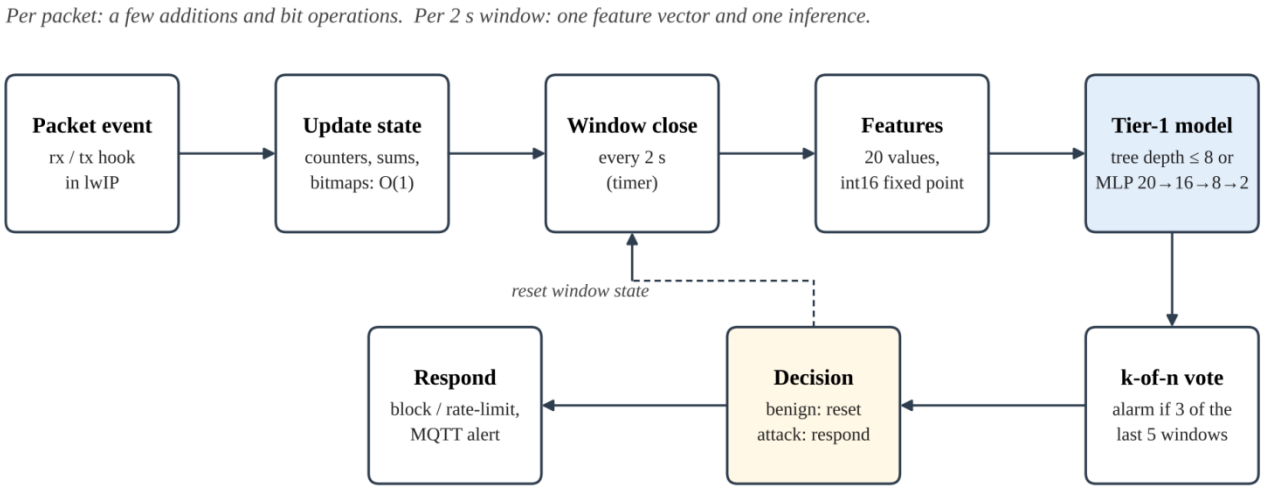


Fig. 2 Per-packet and per-window processing on the device

Table 3 Feature set computed for each 2 s window

No.	Feature	No.	Feature
1	Packets received	11	TCP RST packets
2	Packets sent	12	Share of UDP packets
3	Bytes received	13	ICMP packets
4	Bytes sent	14	Distinct source IP addresses (bitmap estimate)
5	Mean packet size	15	Distinct destination ports (bitmap estimate)
6	Standard deviation of packet size	16	Distinct outgoing destinations (bitmap estimate)
7	Mean inter-arrival time	17	Failed connection or login attempts
8	Standard deviation of inter-arrival time	18	New TCP connections
9	TCP SYN packets received	19	Drop in free heap memory
10	Ratio of SYN to SYN-ACK	20	CPU load of the network task

Means and standard deviations are computed from running sums of values and squares, so no packet history is stored. Distinct counts would normally need a hash table; instead each is estimated with linear counting on a 64-bit bitmap. Each address or port is hashed to one of m = 64 bits, and at the end of the window the number of distinct items n is estimated from the number of bits still zero, V_0 (1):

$$\hat{n} = -m \ln(V_0 / m), \quad m = 64 \tag{1}$$

Three such bitmaps need only 24 bytes and remain accurate up to around a hundred distinct items per window, which is far above benign behaviour for a typical IoT device and well inside the range in which a scan or a botnet join becomes obvious.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

When every bit is set, the feature saturates at its maximum, which itself indicates an attack. All features are converted to 16-bit fixed-point values with per-feature scale factors fixed at training time, so the model runs without floating-point arithmetic.

V. DETECTION MODELS AND DECISION LOGIC

A. *Tier-1 models on the device*

Two tier-1 models are provided so that the IDS can be matched to the device class. The decision tree, trained with scikit-learn [23], is limited to a depth of 8. Each internal node stores a feature index (1 byte), a 16-bit threshold and the index of its right child (2 bytes), so a complete tree of 511 nodes needs at most about 2.6 KB, and one classification takes at most eight comparisons. The tree is the choice for Class 0 and Class 1 devices and is consistent with the very short inference times reported for trees on microcontrollers [14], [17].

The multilayer perceptron has layer sizes 20→16→8→2 with ReLU activations. It has 490 parameters and needs 464 multiply–accumulate (MAC) operations per window. With full INT8 quantization of weights and activations [21] its weights occupy under 0.6 KB including 32-bit biases, and it is run with TensorFlow Lite for Microcontrollers [22] or a hand-written integer kernel. The perceptron draws smoother decision boundaries than a single tree and suits Class 2 devices such as the ESP32-C3 [24]. A random forest [18] of 10 such trees, at up to 26 KB, is kept as an upper reference only.

B. *Voting and false-alarm control*

An IDS that raises an alarm on every flagged window is impractical: at 43,200 windows per day, even a 0.1% per-window false-positive rate p would give 43 false alarms a day. The device therefore raises an alarm only when at least k of the last n windows are flagged. If windows were independent, the probability that a benign stretch triggers an alarm would be (2):

$$P_{FA} = \Sigma_{j=k}^{n} C(n, j)\, p^{j}\, (1 - p)^{n-j} \qquad (2)$$

With k = 3 and n = 5, a confirmed attack must persist for at least three windows, or 6 s, before the device responds. This delay is acceptable for floods, scans and brute-force attempts, which last much longer, and it prevents a single burst of benign traffic, such as a firmware download, from blocking a legitimate server.

C. *Response and training workflow*

On an alarm, the device adds the offending source address to a small block list checked in the packet hook, or limits its packet rate, for a configurable period, and sends an MQTT alert with the last five feature vectors so that the gateway can classify the attack. The device application keeps running throughout.

Fig. 3 shows the training workflow. The central step is that every dataset is replayed, packet by packet, through the same feature code that runs on the device, rather than using the feature columns supplied with the dataset. Models are trained with scikit-learn [23], using class weights to handle the imbalance between benign and attack windows, and features with negligible importance are removed. The trained tree thresholds and perceptron weights are exported as C arrays and installed by a signed over-the-air update.

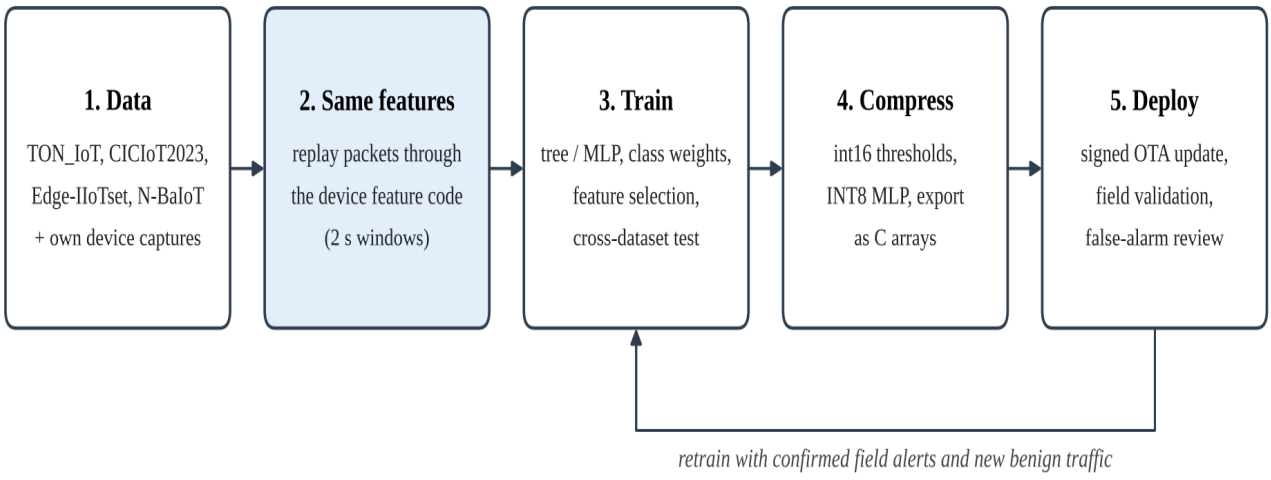


Fig. 3 Training, compression and deployment workflow

©IJRASET: All Rights are Reserved

VI. EXPERIMENTAL METHODOLOGY

A. Datasets and device testbed

Evaluation uses four public datasets — TON_IoT [10], CICIoT2023 [12], Edge-IIoTset [11] and N-BaIoT [6] — replayed through the device feature code as described above. Models are evaluated in two ways: within each dataset, using a split by capture session so that windows from the same session never appear in both training and test sets, and across datasets, training on three and testing on the fourth to measure generalisation.

The complete IDS is then tested on a physical testbed of ESP32-C3 devices running an MQTT sensor application on an isolated Wi-Fi network, with a separate attacker machine. The following attacks are launched against the devices, each repeated several times and interleaved with normal operation and benign traffic bursts such as firmware downloads:

- SYN flood, UDP flood and ICMP flood at several packet rates.
- Port scan and service discovery from the local network.
- Brute-force login attempts against an exposed service.
- Simulated botnet behaviour: the device itself contacting many new destinations.
- Resource exhaustion through many simultaneous connections.

B. Evaluation metrics

Detection is measured per window and per attack episode with precision P, recall R and F1 score (3), where TP, FP and FN are true positives, false positives and false negatives, together with the per-window false-positive rate and the number of false alarms per day after voting:

$$P = TP / (TP + FP), \quad R = TP / (TP + FN), \quad F1 = 2PR / (P + R) \tag{3}$$

Table 4 Performance metrics and targets

Metric	How it is measured	Target
F1 score, within dataset	Session-based split, Eq. (3)	≥ 0.95
F1 score, cross-dataset	Train on three datasets, test on the fourth	Report; ≥ 0.80 desirable
Per-window false-positive rate	Benign windows flagged ÷ benign windows	≤ 0.5%
False alarms per day	Alarms on the testbed during benign operation	≤ 1
Detection delay	Attack start → confirmed alarm	≤ 10 s
Flash and RAM used by IDS	Firmware build map, heap monitor	≤ 10 KB flash, ≤ 2 KB RAM
Per-packet overhead	Cycle counter around the packet hook	Report (µs)
Throughput impact	Application message rate with and without IDS	≤ 5% reduction

VII. RESULTS AND DISCUSSION

A. Memory and computation

Table 5 compares the resource needs of the proposed models, computed from their structure, with values reported for other IDS designs. The window state — roughly 20 accumulators, three 8-byte bitmaps, timestamps and the 40-byte feature vector — fits in under 0.3 KB of RAM, and together with the five-entry vote history and a small block list the module needs under 1 KB of RAM. Both tier-1 models fit within a few kilobytes of flash, so the complete IDS uses only a few per cent of the roughly 100 KiB flash and 10 KiB RAM of a Class 1 device [3]. This is comparable to the node-side footprint of SVELTE [5], but with a learned model instead of fixed rules.

Table 5 Resource requirements: proposed models (computed) and reported values

Model	Flash / model size	RAM	Work per decision	Source
Decision tree, depth ≤ 8	≤ 2.6 KB	< 0.3 KB state	≤ 8 comparisons	Computed
INT8 MLP 20→16→8→2	≈ 0.6 KB (490 params)	< 0.3 KB state	464 MACs	Computed
Random forest, 10 trees (reference)	≤ 26 KB	< 0.3 KB state	≤ 80 comparisons	Computed
SVELTE node modules	1.76 KB ROM	0.365 KB	Rule-based	[5]
TinyML FNN	31 KB	—	—	[16]
INT8 MLP (TON_IoT)	13.03 KB	—	—	[17]
INT8 LightGBM (TON_IoT)	73.85 KB	—	—	[17]

Computation is also small. Every packet costs a few additions, a hash and a bit operation, and every 2 s the device runs at most eight comparisons or 464 MACs. On a 160 MHz ESP32-C3, where a decision tree has been measured at about 5.6 µs [17], the detection work is a negligible fraction of the processor's time; the per-packet overhead and throughput impact are measured in the tests of Section VI.

B. Effect of voting on false alarms

Fig. 4 plots (2) for three decision rules. At a per-window false-positive rate of 0.1%, alarming on every flagged window gives 43 false alarms per day, a 2-of-3 rule about 0.13, and the 3-of-5 rule about 0.0004. Even at a 1% per-window rate, the 3-of-5 rule stays below one false alarm per day (about 0.43). In practice, benign bursts span several consecutive windows, so windows are not independent and the real benefit will be smaller than the curve suggests; the testbed false-alarm count in Table 4 measures it directly.

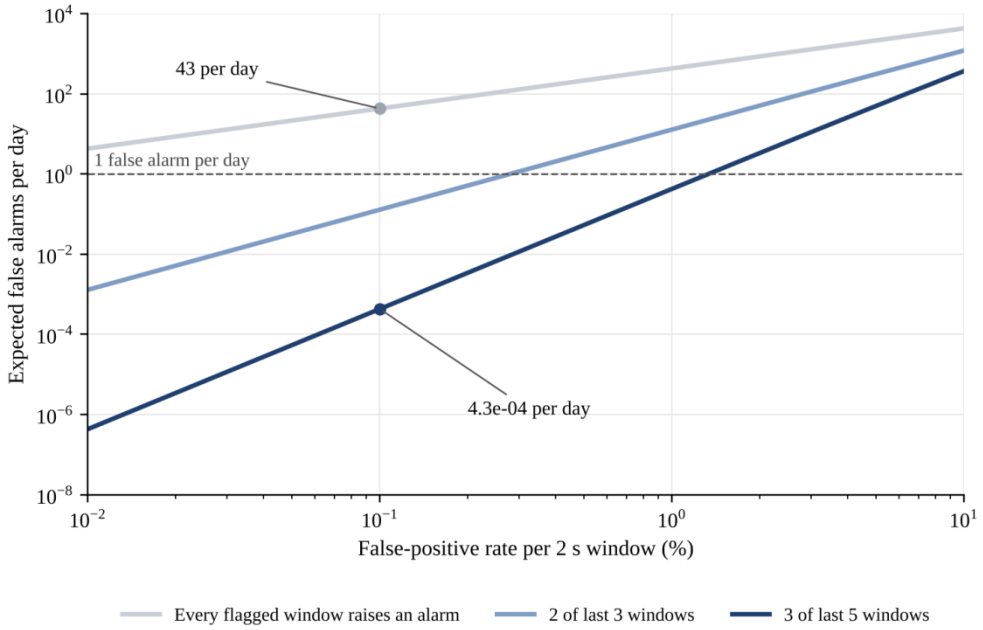


Fig. 4 Expected false alarms per day against per-window false-positive rate (independent windows assumed)

C. Detection results

Detection results are to be entered in Table 6 after the tests in Section VI are completed. Report the number of windows and attack episodes for each case, and all false alarms.

Table 6 Detection results (to be completed after testing)

Test	Decision tree F1	INT8 MLP F1	FPR per window	Notes
TON_IoT (within)	[Insert]	[Insert]	[Insert]	Session split
CICIoT2023 (within)	[Insert]	[Insert]	[Insert]	
Edge-IIoTset (within)	[Insert]	[Insert]	[Insert]	
N-BaIoT (within)	[Insert]	[Insert]	[Insert]	
Cross-dataset (mean of 4)	[Insert]	[Insert]	[Insert]	Leave one dataset out
Testbed, all attacks	[Insert]	[Insert]	[Insert]	Detection delay: [ ] s
Testbed, benign only	—	—	[Insert]	False alarms/day: [ ]
Overhead	—	—	—	Flash [ ] KB, RAM [ ] KB, [ ] µs/packet

D. Discussion

The design deliberately trades some accuracy for deployability. Published IDS models reach F1 scores above 0.99 on their own datasets [16], [17], but those numbers are measured on features the device cannot compute and do not hold across datasets [17].

Defining the features from what the device observes, and replaying all datasets through that code, ensures that what is measured offline is what will run in the field. The k-of-n rule turns a per-window classifier into an alarm with an explicit false-alarm budget, which matters more to users than a fraction of a percent of accuracy.

The design has limitations. A device sees only its own traffic, so attacks that never touch it, or a slow and patient attacker who stays below the window thresholds, will not be detected; the gateway tier and network-level IDSs such as Kitsune [7] remain complementary. An attacker who has already taken full control of the firmware can disable the IDS, so the module protects against network attacks and early compromise rather than a fully owned device; secure boot and signed updates are needed alongside it. The bitmap estimates saturate under heavy scanning, which is acceptable for detection but loses detail for forensics. Finally, all results in Sections 7.1–7.2 are design calculations that must be confirmed by the measurements in Table 6.

VIII. CONCLUSION AND FUTURE SCOPE

This paper presented a lightweight machine-learning intrusion detection system for resource-constrained IoT devices. A packet hook updates fixed-size state in constant time, 20 features are computed every 2 s, and a depth-limited decision tree or an INT8 perceptron of 490 parameters classifies each window, with a 3-of-5 vote controlling false alarms and a local block-and-alert response. The design needs at most a few kilobytes of flash and under 1 KB of RAM, within the budget of a Class 1 constrained device. A review of 16 studies showed that this focus on features computable on the device, small models and cross-dataset evaluation addresses the main obstacles to deploying machine-learning IDS on IoT hardware.

Future work will complete the dataset and testbed evaluation, add on-device adaptation to each device's normal traffic, explore federated training across many devices using Edge-IIoTset-style data [11], and extend the feature set to IPv6, 6LoWPAN and Bluetooth Low Energy traffic.

IX. ACKNOWLEDGMENT

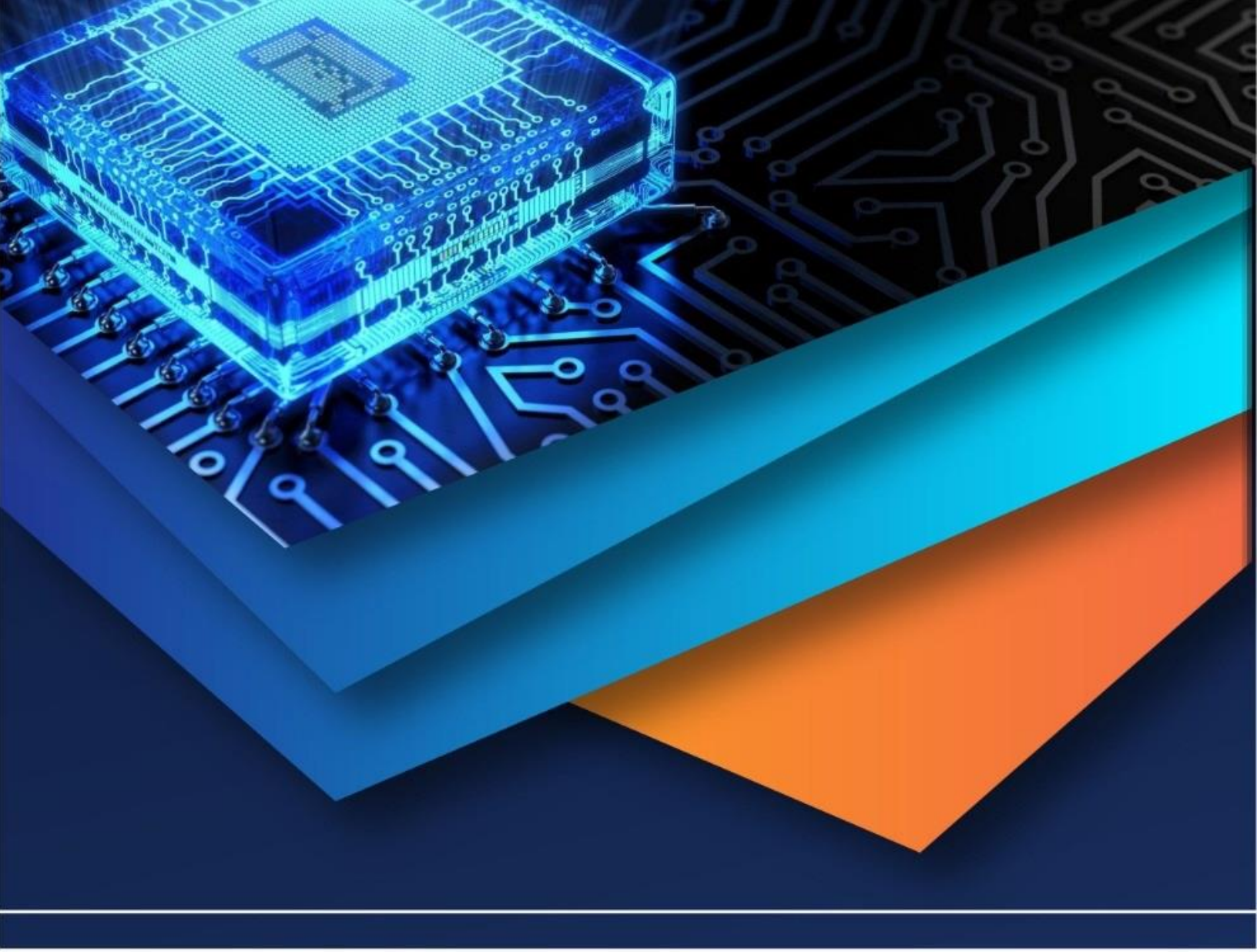
[Insert acknowledgment of sponsors, institutions or individuals who supported this work.]

REFERENCES

[1] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, “DDoS in the IoT: Mirai and other botnets,” Computer, vol. 50, no. 7, pp. 80–84, 2017.
[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein et al., “Understanding the Mirai botnet,” in Proc. 26th USENIX Security Symposium, 2017, pp. 1093–1110.
[3] C. Bormann, M. Ersue, and A. Keränen, “Terminology for constrained-node networks,” IETF RFC 7228, May 2014.
[4] K. A. P. da Costa, J. P. Papa, C. O. Lisboa, R. Munoz, and V. H. C. de Albuquerque, “Internet of Things: A survey on machine learning-based intrusion detection approaches,” Computer Networks, vol. 151, pp. 147–157, 2019, doi: 10.1016/j.comnet.2019.01.023.
[5] S. Raza, L. Wallgren, and T. Voigt, “SVELTE: Real-time intrusion detection in the Internet of Things,” Ad Hoc Networks, vol. 11, pp. 2661–2674, 2013, doi: 10.1016/j.adhoc.2013.04.014.
[6] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, “N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders,” IEEE Pervasive Computing, vol. 17, no. 3, pp. 12–22, 2018.
[7] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, “Kitsune: An ensemble of autoencoders for online network intrusion detection,” in Proc. Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 2018.
[8] R. Doshi, N. Apthorpe, and N. Feamster, “Machine learning DDoS detection for consumer Internet of Things devices,” in Proc. IEEE Security and Privacy Workshops (SPW), 2018, pp. 29–35, doi: 10.1109/SPW.2018.00013.
[9] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, “Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset,” arXiv:1811.00701, 2018.
[10] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar, “TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems,” IEEE Access, vol. 8, pp. 165130–165150, 2020, doi: 10.1109/ACCESS.2020.3022862.
[11] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, “Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning,” IEEE Access, vol. 10, pp. 40281–40306, 2022.
[12] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, “CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment,” Sensors, vol. 23, no. 13, Art. no. 5941, 2023, doi: 10.3390/s23135941.
[13] Z. Alwaisi, T. Kumar, E. Harjula, and S. Soderi, “Securing constrained IoT systems: A lightweight machine learning approach for anomaly detection and prevention,” Internet of Things, vol. 28, Art. no. 101398, 2024, doi: 10.1016/j.iot.2024.101398.
[14] A. Javed, A. Ehtsham, M. Jawad, M. N. Awais, A.-u.-H. Qureshi, and H. Larijani, “Implementation of lightweight machine learning-based intrusion detection system on IoT devices of smart homes,” Future Internet, vol. 16, no. 6, Art. no. 200, 2024, doi: 10.3390/fi16060200.
[15] P. Fusco, G. P. Rimoli, and M. Ficco, “TinyIDS – An IoT intrusion detection system by tiny machine learning,” in Lecture Notes in Computer Science, vol. 14816, Springer, 2024, pp. 71–82, doi: 10.1007/978-3-031-65223-3_5.
[16] J. Lundqvist, A. Hadzic, T. M. Kirkeluten, H. Pedersen, J. Holth, M. H. Johansson, and M. P. N. Halkjelsvik, “Lightweight machine learning models for intrusion detection on IoT devices,” in Proc. Norsk IKT-konferanse for forskning og utdanning (NIKT), vol. 37, no. 3, 2025, doi: 10.5324/jrxdjb92.

[17] E. P. De Bernardis, O. Kuznetsov, M. Arnesano, P. Zhansaya, and M. Sydykova, "Lightweight machine learning intrusion detection for IoT/IIoT networks: Quantisation strategies and physical deployment on resource-constrained microcontrollers," Electronics, no. 13, Art. no. 2869, 2026, doi: 10.3390/electronics15132869.
[18] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
[19] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.
[20] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017.
[21] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 2704–2713.
[22] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries et al., "TensorFlow Lite Micro: Embedded machine learning for TinyML systems," in Proc. Machine Learning and Systems (MLSys), vol. 3, pp. 800–811, 2021.
[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
[24] Espressif Systems, "ESP32-C3 series datasheet," technical documentation.
[25] OASIS, "MQTT version 3.1.1," OASIS Standard, A. Banks and R. Gupta, Eds., Oct. 2014.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)