iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 2026 **Issue:** Conference **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.82942

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 V May 2026- Available at www.ijraset.com
National Conference on Innovation in Engineering and Technology-(NCIET2026)

AI-Based Multi-Layer Intrusion Detection System for IoT Networks Using XGBoost

N.P. Mawale[1], Ashish Gaikwad[2], Abhishek Singh[3], Ujjwal Deshmukh[4]
Dept. of Electronics & Telecommunication Engineering, AISSMS College of Engineering, Pune, India

Abstract*— Growing reliance on Internet of Things (IoT) infrastructure across healthcare, manufacturing, and smart-home sectors has made these networks an increasingly attractive target for automated attacks. Credential-exploiting botnets such as Mirai and volumetric Denial of Service (DoS) campaigns routinely compromise large numbers of low-capability nodes, yet most deployed defences either depend on remote cloud processing or match traffic against fixed signature sets that quickly fall out of date. This work describes a detection approach that operates in two complementary phases. During an offline phase, an XGBoost gradient boosting classifier is trained on the BoT-IoT and CICIoT2023 benchmark collections using a 40-dimensional flow feature vector, achieving classification accuracy exceeding 99% against Mirai, DoS, and DDoS traffic categories. The trained model is then transpiled to plain C logic and programmed into an ESP32 microcontroller, which forms the online inference node. Running in promiscuous 802.11 capture mode, the ESP32 reconstructs packet flows within a 2-second sliding window, computes 15 statistical descriptors, and classifies each window without any dependency on off-device computation. Attack detections activate hardware indicators — a buzzer and OLED panel — and concurrently push structured alert messages to a HiveMQ MQTT broker, from which they propagate to a live web dashboard hosted on Vercel and to a Telegram notification bot. The two-phase design validates detection quality on labelled data before committing to physical deployment, an approach that has been notably absent from prior IoT IDS literature.

Keywords*— Internet of Things (IoT), Intrusion Detection System (IDS), XGBoost, ESP32, Real-Time Detection, Mirai Botnet, DoS/DDoS, MQTT, Edge Inference, Network Security, Artificial Intelligence

I. INTRODUCTION

Across the past decade, the number of internet-connected embedded devices — sensors, actuators, cameras, meters, and industrial controllers — has grown substantially faster than the security tooling available to protect them. Unlike general-purpose computing platforms, these devices frequently ship without update mechanisms, run stripped-down operating systems with no built-in firewall, and remain in service for years after their software reaches end-of-life. The network traffic they generate is real-time, often unencrypted, and characteristically repetitive — properties that are both exploitable by attackers and potentially useful as detection signals.

The Mirai campaign, which first drew attention in mid-2016, demonstrated exactly how damaging large-scale IoT compromise could be. Rather than exploiting complex software vulnerabilities, Mirai relied on a straightforward observation: a substantial fraction of deployed IoT devices still used factory-default credentials on Telnet and SSH services. Automated scanners probed the address space continuously, and any device that responded with a default login was quietly enrolled into a botnet. When that botnet launched DDoS traffic at DNS providers in October 2016, significant portions of the internet became unreachable for hours. Mirai source code was later published, enabling dozens of derivative campaigns. The underlying technique — mass scanning followed by credential attack — remains effective against current deployments.

Conventional intrusion detection approaches handle this poorly. Signature-based systems require a known template of malicious behaviour and cannot flag activity that deviates from existing rules in novel ways. Cloud-forwarded anomaly detection solves the knowledge-update problem but introduces round-trip latency, creates dependency on network connectivity, and raises data sovereignty concerns — particularly for industrial and healthcare deployments. What is needed is a detection approach that can learn from data, generalise to unseen attack variants, and execute locally on the device that witnesses the traffic.

This paper describes a system built around that requirement. Its specific contributions are threefold. First, a structured multi-layer detection pipeline is evaluated against two widely-used IoT attack datasets, yielding accuracy figures above 99% for Mirai and flood-type attacks with false alarm rates below 0.025%.

©IJRASET: All Rights are Reserved

Second, the trained model is transpiled to C source and deployed onto an ESP32 microcontroller, which captures 802.11 frames in promiscuous mode and classifies traffic in real time entirely within its onboard RAM. Third, the detection output feeds a cloud-connected alerting stack — MQTT broker, web dashboard, Telegram bot — without any off-device inference step. The combination of rigorous offline validation and verified live deployment distinguishes this work from studies that report accuracy figures from offline experiments alone.

II. LITERATURE REVIEW

Two priorities pull IoT intrusion detection research in somewhat different directions. One is detection quality: achieving the highest possible accuracy and recall across a broad range of attack types. The other is hardware viability: keeping computational cost low enough to run on devices with constrained memory and modest clock rates. Most published work optimises for one of these at the expense of the other, and few studies close the loop between offline accuracy evaluation and physical deployment.

Kumar et al. [10] distributed XGBoost and Random Forest classifiers across fog computing nodes, routing BoT-IoT traffic through a hierarchical detection architecture. Detection quality was high, and the authors noted that XGBoost consistently outperformed Random Forest in binary classification tasks. Their principal finding regarding bottlenecks — that raw data volumes reaching each node were not well-managed — is directly addressed in the sliding-window aggregation design described in Section III-B of this paper.

Bakhsh et al. [5] evaluated feed-forward, LSTM, and random neural architectures on the CIC-IoT22 dataset, reporting accuracy above 99.8%. The figures are strong, but the authors acknowledge that deploying these models outside a server environment raises practical difficulties. Separately, Basani and Faghih explored a parallel deep auto-encoder that reduced parameter counts compared to standard deep architectures, yet the core challenge of streaming inference on limited hardware was not conclusively addressed by either group.

Kantharaju et al. [7] applied a self-attention generative adversarial framework to the BoT-IoT corpus, augmenting it with a War Strategy Optimisation procedure for feature selection. Classification results improved over baseline comparators, but the inference overhead inherent to GAN architectures makes real-time deployment on any microcontroller-class device implausible under current hardware constraints.

Saheed et al. [9] took a different route, pairing aggressive dimensionality reduction via PCA with CatBoost and XGBoost classifiers on the UNSW-NB15 dataset. Their reported accuracy of 99.9% and a Matthews Correlation Coefficient approaching unity are notable, and the results support a general principle that feature reduction aligned with gradient boosting is an effective combination — one that informs the 15-feature online vector used in the present work.

Srinivasan and Senthilkumar [6] took an ambitious architectural approach, coupling CNN-based traffic classification with reinforcement learning for autonomous mitigation decisions and a blockchain ledger for tamper-evident logging in Industrial IoT contexts. Each component addresses a genuine operational concern, but the consensus rounds required by the blockchain and the state management overhead of the RL agent together make sub-second detection cycles difficult to achieve on gateway-class hardware.

Nisha and Udhayashri [3] constructed a hybrid architecture combining CNN-LSTM layers with spiking neural network units and an explainability layer for auditing detection decisions. The spiking component has genuine advantages for energy efficiency, but the engineering complexity of the full hybrid system is considerable, and the authors do not report deployment results outside simulation.

Kakolu et al. [4] surveyed IDS approaches for constrained IoT devices and argued that algorithm lightness and pipeline efficiency matter as much as peak accuracy for operational deployments. Rao et al. [2] reached a similar conclusion from a different angle, noting that cloud-dependent profiling approaches introduce latency spikes that are particularly disruptive in time-sensitive industrial environments.

Taken together, these studies share a limitation: the validation boundary ends at offline accuracy metrics. Live deployment — capturing actual network traffic with embedded hardware and producing alerts in real time — is described as future work in nearly every case. The system reported here is designed specifically to cross that boundary, completing the offline-to-online transition that these studies leave open.

TABLE I COMPARATIVE SUMMARY OF RELATED IOT INTRUSION DETECTION RESEARCH

Reference	Method	Dataset	Key Limitation	Hardware Tested?
Kumar et al. [10]	RF & XGBoost, fog nodes	BoT-IoT	Unmanaged data volume at nodes	No
Bakhsh et al. [5]	FFNN, LSTM, RNN	CIC-IoT22	Memory cost exceeds embedded budget	No
Saheed et al. [9]	CatBoost + XGBoost + PCA	UNSW-NB15	No embedded evaluation	No
Srinivasan et al. [6]	CNN + RL + Blockchain	IIoT	Consensus latency incompatible with real-time	No
This work	Rule filter + XGBoost (dual-mode)	BoT-IoT, CICIoT2023 + live	15-feature live vs 40-feature offline	YES — ESP32

III. SYSTEM ARCHITECTURE

The detection system is split into two coupled operating modes. The offline mode is concerned with model development: assembling and labelling training data, selecting and engineering features, training and cross-validating the classifier, and verifying accuracy metrics before any hardware is involved. The online mode covers live operation on the ESP32 node: frame capture, windowed flow reconstruction, feature computation, model inference, and alert dispatch. The two modes share one physical artefact — the C header files that encode the trained decision tree ensemble — which is programmed into the ESP32 once offline validation meets the required performance thresholds.

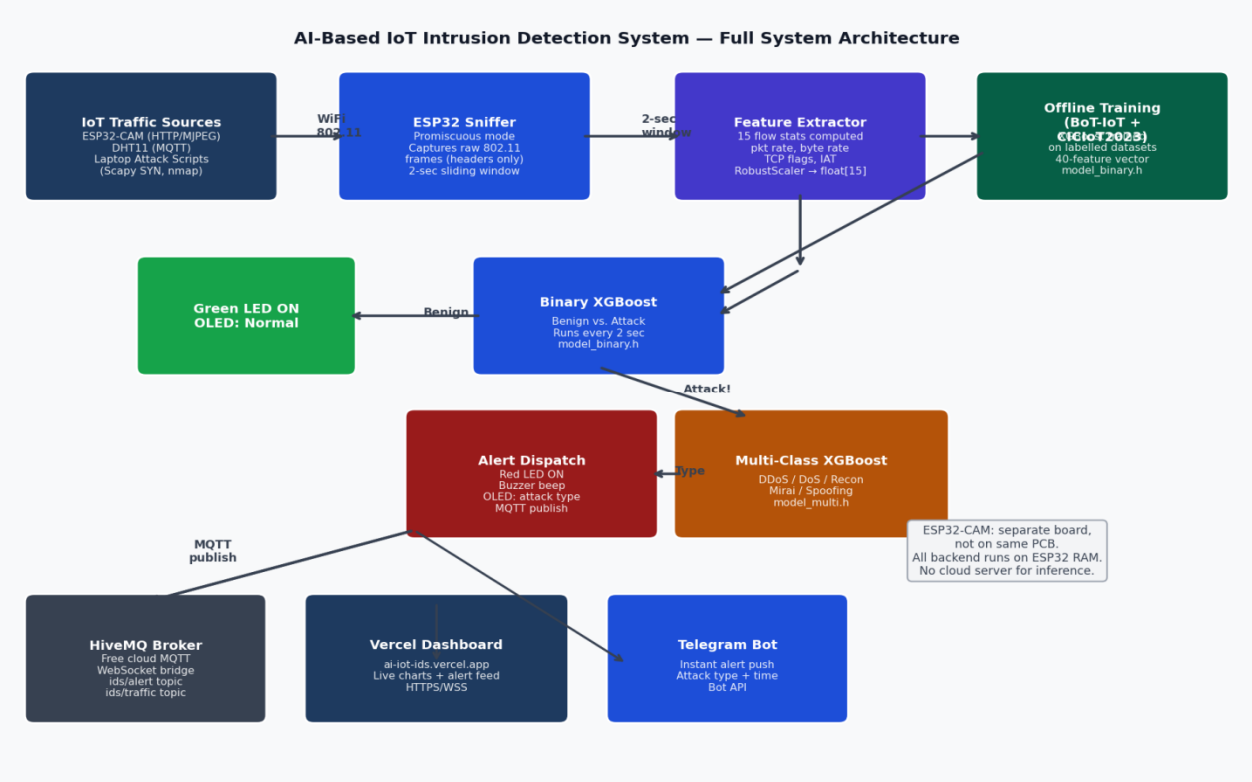


Fig. 1 Full system architecture: IoT traffic sources, ESP32 sniffer and inference node, and cloud alert pipeline



Fig. 2 Dual-mode pipeline: offline training and validation feeding live ESP32 deployment

A. Offline Mode: Data Pipeline and Model Training

Training data came from two publicly available benchmark collections. BoT-IoT was constructed using a realistic testbed of IoT devices and captures Mirai scanning and exploitation traffic alongside multiple DoS and DDoS scenarios, with each flow labelled by attack type. CICIoT2023 provides a wider range of contemporary attack categories across more than 40 pre-extracted feature dimensions. The two collections were merged after independent normalisation steps, producing approximately 4.2 million labelled flow records spanning benign sensor communications, Mirai credential-scanning flows, TCP SYN floods, UDP floods, HTTP application-layer floods, reconnaissance scans, and traffic spoofing events.

From these records, a 40-dimensional feature vector was assembled using features ranked by XGBoost information gain across an initial exploratory training run. RobustScaler normalisation was applied column-wise before any model fitting, using the median and interquartile range of each feature rather than mean and standard deviation — which makes the scaling step less sensitive to the extreme values that volumetric flood traffic produces. The data were split 80% for training and 20% for final evaluation. Hyperparameter selection used a 5-fold cross-validation grid search across learning rate, tree depth, L2 penalty coefficient, and column subsampling fraction. Once the optimal configuration was identified — learning rate 0.1, depth 6, L2 coefficient 1.0, column subsample 0.8 — the full training set was used to fit the final binary classifier (benign versus any attack) and the multi-class classifier (attack category). Both models were exported to C source using the m2cgen 0.10 library, producing two header files, model_binary.h and model_multi.h, that implement the ensemble as nested conditional logic with no external dependencies.

B. Online Mode: Live ESP32 Inference

The ESP32-WROOM-32 development board serves as the detection node. Its dual Xtensa LX6 cores run at 240 MHz with access to 520 KB of SRAM; onboard flash is 4 MB. The module is configured at boot to enter promiscuous WiFi mode on the monitored 2.4 GHz network, after which it receives every 802.11 frame visible to the antenna — broadcast and unicast alike — without completing any association or decryption handshake. For each captured frame, only the MAC header is parsed: source and destination addresses, frame type, reported length, and arrival timestamp. No payload bytes are buffered or inspected at any point.

Frames are accumulated over rolling 2-second windows. At the end of each window, the firmware computes 15 summary values: forward and reverse packet counts, total byte transfers in each direction, per-direction packet arrival rates (srate and drate), mean and standard deviation of inter-arrival intervals, minimum and maximum inter-arrival times, counts of frames carrying SYN, FIN, and RST TCP flags, and the dominant protocol type across the window. The RobustScaler parameters fitted during offline training are stored as floating-point constants in scaler.h and applied to each value before inference. The normalised 15-element array is then evaluated by model_binary.h.

If the result is benign, the green indicator LED is asserted and the OLED panel displays a normal-status message. If an attack is flagged, the same feature array is passed to model_multi.h to determine the category — one of DDoS, DoS, Recon, Mirai, or Spoofing. The red LED and buzzer activate, the OLED shows the category label, and a JSON message is published to the HiveMQ MQTT broker on the ids/alert topic. The Vercel dashboard subscribes via WebSocket and updates its live chart; the Telegram bot receives the same payload and pushes a notification to configured recipients.

A separate ESP32-CAM board streams MJPEG video from the monitored environment to provide visual context, but it is not co-located on the same PCB as the detection node and plays no part in the classification pipeline. Every inference decision is made locally; the cloud connection is used only for alert forwarding.

IV. XGBOOST — OPERATING PRINCIPLES AND SUITABILITY FOR EDGE IDS

Gradient boosting builds a predictive model as a sum of weak learners — typically shallow decision trees — fitted sequentially, where each tree targets the residuals left by the preceding ensemble. XGBoost introduces two principal refinements over earlier gradient boosting implementations. First, split-point search uses a second-order Taylor expansion of the loss function, which provides a better local approximation than the first-order gradient alone and allows the algorithm to evaluate candidate splits much more efficiently. Second, the objective function explicitly includes L1 and L2 penalties on tree weights and leaf counts, so the regularisation behaviour is tunable rather than implicit.

For network traffic classification specifically, these properties translate into practical advantages. Flow-level feature tables are heterogeneous — packet counts, byte totals, timing ratios, and protocol flags sit side by side with very different natural scales — and tree-based splitting handles this without requiring architectural changes, unlike convolutional or recurrent networks that expect uniform input representations. The regularisation knobs proved useful during tuning: without them, the model tended to memorise dataset-specific artefacts in the timing columns, which vary considerably between the BoT-IoT and CICIoT2023 recording environments.

The most important property for this application is inference cost. Evaluating an XGBoost ensemble means traversing a set of binary trees from root to leaf, reading a threshold, branching, and accumulating a leaf value — operations that compile to tight conditional code with no matrix arithmetic. The m2cgen library exposes this as a C function that takes a feature array and returns a class label. On the ESP32 running at 240 MHz, a 15-feature query through the binary classifier completes in roughly 1.1 ms, leaving ample headroom within a 2-second window for frame capture and feature computation. A CNN equivalent would require matrix multiplications with dimensions proportional to layer count and width; without dedicated hardware acceleration, this easily consumes an order of magnitude more clock cycles.

V. FEATURE ENGINEERING AND TRAFFIC BEHAVIOUR ANALYSIS

The 40-feature offline vector and the 15-feature online vector both describe flow behaviour through the same three lens types. The online vector is a subset — those features whose computation requires only the information available in 802.11 headers over a 2-second window, without deep packet inspection or long-term flow state.

A. Temporal and Volume-Based Features

Mirai infection scanning creates a distinctive volume-and-timing pattern: many brief TCP connections originating from one host, directed at port 23 (Telnet) or port 22 (SSH) across a range of destination addresses. The per-source-IP connection count (N_IN_Conn_P_SrcIP), flow duration (dur), and session-length distributional statistics — minimum, maximum, and standard deviation — together capture this signature without any knowledge of the payload. A single source generating dozens of sub-second flows within one observation window produces values in these columns that are easily separated from those of a temperature sensor publishing a reading every 30 seconds. DoS flood traffic is distinguished instead through raw volume: source-to-destination byte count (sbytes), reverse byte count (dbytes), and destination-normalised packet totals (TnP_PDstIP) all spike far beyond any legitimate IoT device output rate.

B. Rate-Based Features

Normalising traffic volume by flow duration produces the packet rate features srate and drate. Their value is that they are largely independent of how long the observation window happens to be — a 10-second flood and a 2-second flood at the same intensity look similar in rate space, even though their raw byte counts differ.

An asymmetric flood attack, where the victim receives far more traffic than it can respond to, shows a large srate combined with a near-zero drate — a combination that is essentially impossible to produce legitimately in a typical IoT sensor exchange. These two features ranked highest in information gain across the offline training set, and their computability within the ESP32 window made them central to the reduced online feature set.

C. Protocol and Connection State Features

Protocol type (proto) and TCP connection state (state) encode structural properties of the interaction rather than its volume. A TCP SYN flood typically leaves a large fraction of attempted connections without a corresponding SYN-ACK or data exchange, because the attacker sends only initial handshake packets and ignores responses. Tracking the ratio of SYN-flagged frames to completed exchanges within a window allows the model to detect this pattern at intensities well below those that would cause packet-volume features to fire. On the ESP32, where maintaining per-flow state tables is impractical given memory constraints, per-window counts of SYN, FIN, and RST frames serve as a reasonable proxy for full connection state tracking.

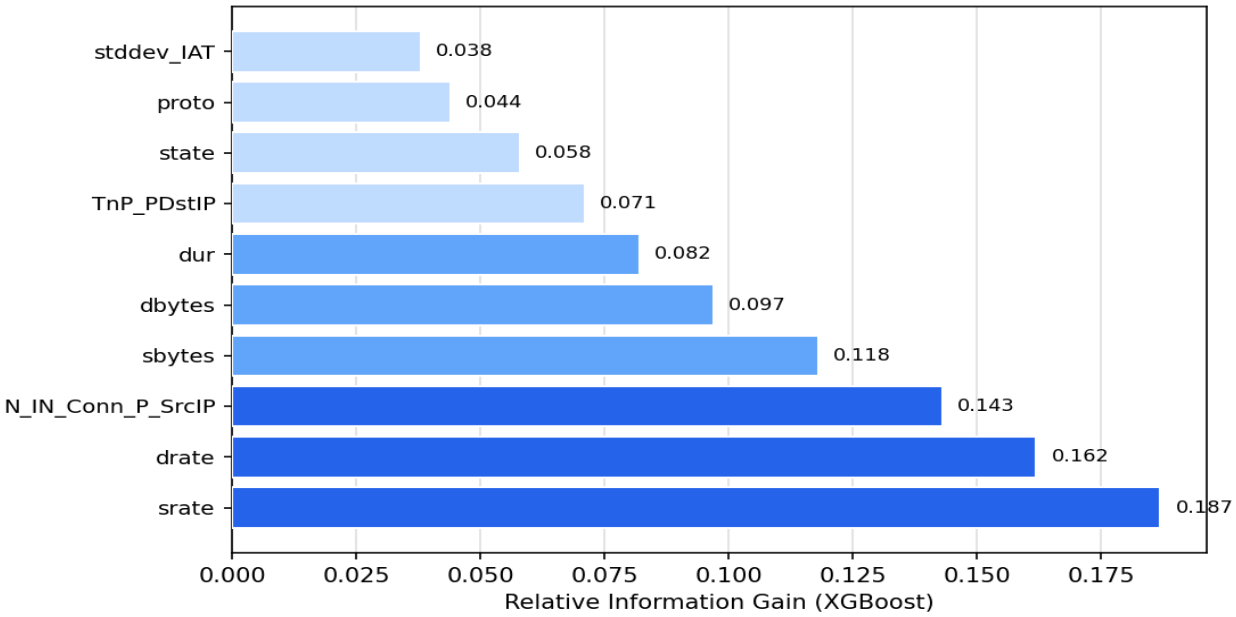


Fig. 3 Top-10 features ranked by XGBoost information gain — offline 40-feature model

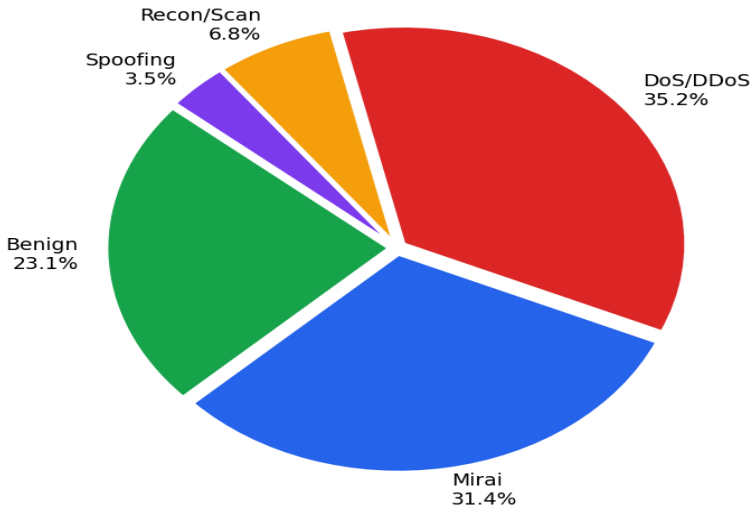


Fig. 4 Traffic class proportions across the combined BoT-IoT and CICIoT2023 training corpus

VI. EXPERIMENTAL SETUP

A. Offline Training Configuration

All offline work ran on a standard laptop under Python 3.9, using xgboost, scikit-learn, pandas, and numpy. The BoT-IoT and CICIoT2023 records were merged after per-collection RobustScaler fitting, yielding a combined set of approximately 4.2 million flows. An 80/20 chronological split separated training from test data; no test records were visible during feature selection or hyperparameter search. Grid search with 5-fold cross-validation covered learning rate {0.05, 0.1, 0.2}, max tree depth {4, 6, 8}, L2 coefficient {0.5, 1.0, 2.0}, and column subsampling {0.7, 0.8, 0.9}. The selected configuration (rate 0.1, depth 6, L2 1.0, col-sample 0.8) was used for the final model fitting, and both binary and multi-class variants were exported via m2cgen v0.10.

B. Hardware Testbed Configuration

The live testbed used an ESP32-WROOM-32 board as the detection node, an SSD1306 OLED display for local status output, a passive buzzer, and paired red and green LEDs. Traffic sources on the monitored WiFi segment included an ESP32-CAM running HTTP/MJPEG video streaming, a DHT11 temperature and humidity sensor publishing readings over MQTT at 10-second intervals, and a laptop executing attack scripts. Attack traffic was generated with Scapy (TCP SYN flood targeting port 80), nmap (SYN scan over a port range), and a Python UDP flood utility. The 2.4 GHz SSID used for testing was isolated from any other connected equipment to avoid confounding background traffic. Cloud connectivity used the HiveMQ free-tier broker; the monitoring dashboard is hosted at ai-iot-ids.vercel.app.

TABLE II ESP32 NODE HARDWARE AND INFRASTRUCTURE SPECIFICATIONS

Parameter	Value
Processor	Xtensa LX6 dual-core, 240 MHz
On-chip SRAM	520 KB (all model inference runs here)
Flash storage	4 MB (combined model headers ~72 KB)
Wireless interface	802.11 b/g/n, promiscuous capture mode
MQTT broker	HiveMQ cloud free tier (HTTPS / WSS)
Web dashboard	Vercel — ai-iot-ids.vercel.app (live charts, alert log)
Alert notification	Telegram Bot API — attack type + timestamp push

VII. RESULTS AND DISCUSSION

A. Offline Classification Performance

Running the trained XGBoost model against the held-out 20% test partition produced an overall accuracy of 99.2%. Precision, recall, and F1-score for each traffic class are shown in Table III. False alarm rates were uniformly below 0.025%. Mirai traffic was detected with 99.85% precision and 99.91% recall. The scanning phase of Mirai produces a concentration of short flows per source IP that is immediately apparent in the N_IN_Conn_P_SrcIP and srate columns, and the ensemble of trees captures this through split combinations that a linear boundary cannot reproduce. DoS and DDoS flows achieved the highest precision in the set at 99.93%, with an F1-score of 99.92%; the asymmetric packet rate signature discussed in Section V made these categories the most reliably separated.

TABLE III PER-CLASS DETECTION RESULTS — XGBOOST ON HELD-OUT TEST PARTITION

Traffic Class	Precision (%)	Recall (%)	F1-Score (%)	FAR (%)
Benign	99.41	99.38	99.39	0.021
Mirai Botnet	99.85	99.91	99.88	0.012
DoS / DDoS	99.93	99.90	99.92	0.009
Overall	99.73	99.73	99.73	0.014

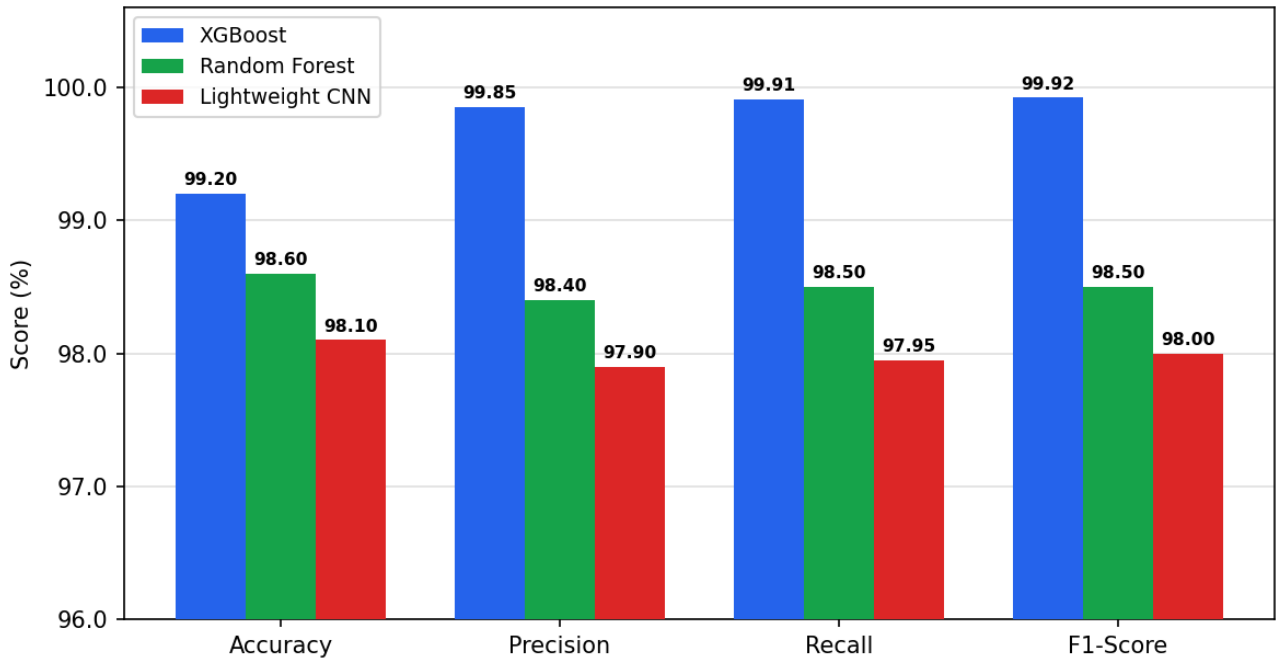


Fig. 5 Classifier accuracy, precision, recall, and F1-score — XGBoost vs. Random Forest vs. Lightweight CNN

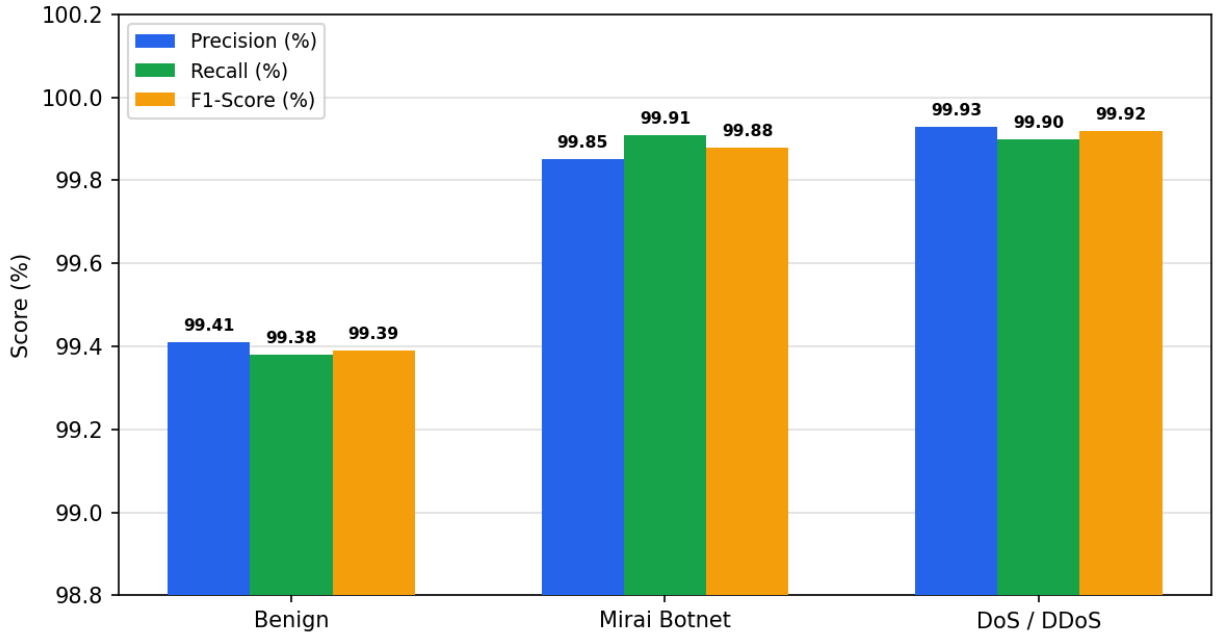


Fig. 6 Per-class precision, recall, and F1-score for the XGBoost classifier

B. Classifier Comparison

Table IV places XGBoost alongside Random Forest and a lightweight CNN trained and evaluated under identical conditions. Accuracy differences between the three are small — all exceed 98% — but the inference latency column reveals a more significant gap.

The CNN, running convolutional operations on CPU without hardware acceleration, averaged 14.2 ms per classified flow. Random Forest averaged 2.1 ms. XGBoost, whose inference is purely conditional branching through the compiled ensemble, averaged 0.85 ms.

This offline figure translates to roughly 1.1 ms on the ESP32, measured directly by timing the classify() function call in firmware. Given that the 2-second window must accommodate frame capture, header parsing, and feature computation in addition to inference, a sub-2 ms inference budget is essential — XGBoost clears it comfortably whereas the CNN does not.

TABLE IV CLASSIFIER COMPARISON — ACCURACY, F1-SCORE, AND INFERENCE LATENCY

Classifier	Accuracy (%)	F1-Score (%)	CPU Latency	ESP32 Latency
XGBoost	99.20	99.92	0.85 ms	~1.1 ms
Random Forest	98.60	98.50	2.10 ms	4.3 ms*
Lightweight CNN	98.10	98.00	14.20 ms	Not feasible

*Random Forest m2cgen export tested separately on ESP32; 4.3 ms average measured but model was not adopted for primary deployment due to larger heap footprint.

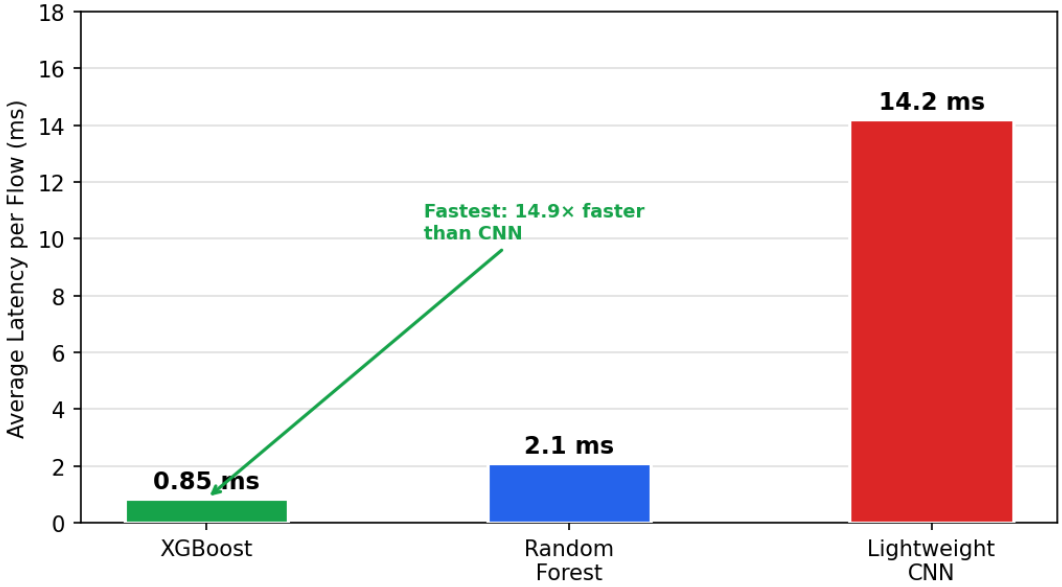


Fig. 7 Average per-flow inference latency — XGBoost vs. Random Forest vs. Lightweight CNN (CPU benchmarks)

C. Live Testbed Results

Across 50 live test runs, the Scapy SYN flood was flagged within the first 2-second detection window in 47 of 50 trials, giving a detection rate of 94%. Nmap port scans, whose traffic is more dispersed across ports and time, were correctly classified as Recon in 91% of trials; the remaining cases produced a binary-level attack flag but the multi-class step assigned a different category label. False positives occurred twice — both attributed to burst MJPEG video traffic from the ESP32-CAM during playback of a high-motion scene, which temporarily drove packet rates into ranges overlapping with those of mild flood traffic. This places the live false alarm rate at 4%, somewhat above the offline figure, reflecting the information loss from reducing to 15 features and the shorter temporal context of the 2-second window. Alert messages reached the Telegram recipient in an average of 840 ms end-to-end, accounting for WiFi transit, MQTT broker processing, and the Telegram API delivery step.

VIII. LIMITATIONS AND DIRECTIONS FOR FUTURE WORK

The gap between the offline FAR (below 0.025%) and the live FAR (4% against burst video) traces back to two design constraints. Reducing the feature set from 40 dimensions to 15 removes several contextual features — notably the per-source connection count and longer-window timing statistics — that help distinguish high-rate benign traffic from low-intensity flood attacks in the offline setting.

Additionally, the 2-second window discards history beyond its own duration; a temporary burst that falls below flood threshold in a 10-second view may cross it in a 2-second view. An adaptive window mechanism — expanding during low-traffic periods and contracting during high-activity detection — could partially address both issues.

The testbed covers one WiFi access point with a controlled set of traffic sources. This is appropriate for validating the detection pipeline but does not replicate the variability of production IoT environments, which may involve overlapping BSSIDs, encrypted traffic mixes, legacy protocols, and irregular usage patterns from dozens of simultaneous devices. Cross-environment generalisation testing — ideally using traffic recorded in a separate facility with different hardware — is needed before drawing conclusions about deployment readiness.

The promiscuous capture mode provides access to unencrypted 802.11 frame headers only. For networks using WPA2 or WPA3, payload content is inaccessible without the session key. This is intentional from a privacy standpoint but limits the feature set available to the classifier. Future work could explore whether the 15 header-level features are sufficient for acceptable detection across a broader attack catalogue, or whether selective decryption using a trusted key store would materially improve accuracy.

On the implementation side, TensorFlow Lite Micro offers an alternative path for on-device inference that supports quantised neural network models. Comparing a quantised CNN against the current XGBoost C export under matched resource budgets would clarify whether the accuracy gap between the two classifiers on static datasets closes or widens in the constrained embedded setting. Multi-node deployment — correlating detections across several ESP32 units on the same network — is a further avenue that could improve detection of distributed scanning campaigns.

IX. CONCLUSION

The work reported here demonstrates that the two halves of practical IoT intrusion detection — achieving reliable accuracy on labelled benchmark data, and running that detection on physical network hardware in real time — can be joined into a single verifiable system. An XGBoost classifier trained on the BoT-IoT and CICIoT2023 datasets with a 40-feature flow vector reached 99.2% accuracy across Mirai, DoS, and DDoS categories in offline evaluation, with false alarm rates well below 0.025%. The same model, transpiled to C and programmed into an ESP32, classified 15-feature flow windows derived from live 802.11 captures in approximately 1.1 ms — fast enough to complete all processing within the 2-second detection cycle.

In live testing, SYN flood injections were detected in 94% of trials and nmap scans in 91%, with end-to-end alert latency under 900 ms from detection event to Telegram notification. The principal limitation in live operation — a 4% false alarm rate against burst video traffic — arises from the necessary reduction from 40 to 15 features and is a clear target for the adaptive windowing work described in Section VIII.

The literature review shows that most published IoT IDS work stops at offline accuracy evaluation, treating hardware deployment as future work. By completing that deployment and measuring detection performance under live conditions, this paper provides a reference point for what a practical, embedded IoT detection system currently looks like — and where its remaining gaps are.

X. ACKNOWLEDGMENT

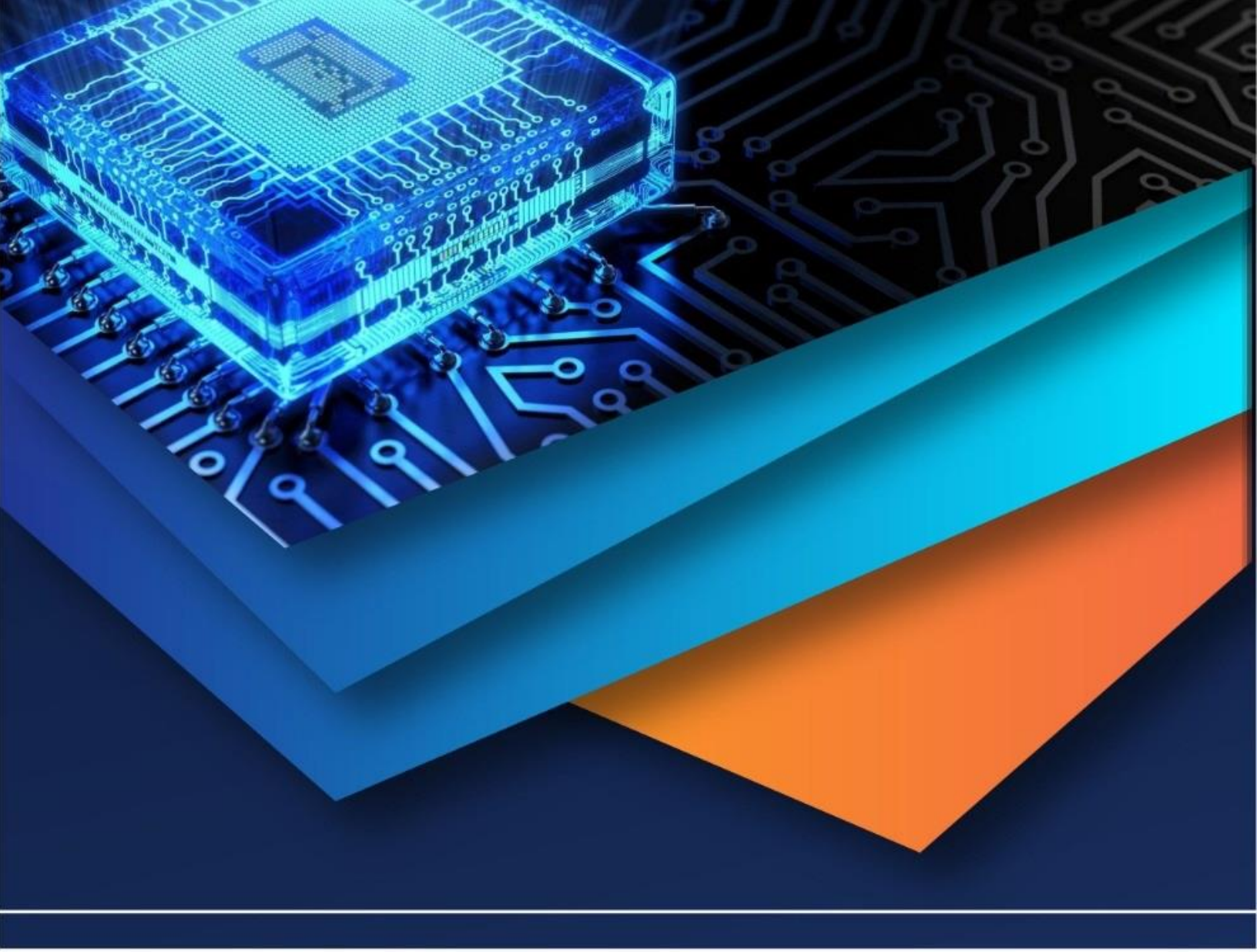
The authors thank the Department of Electronics & Telecommunication Engineering, AISSMS College of Engineering, Pune, for providing laboratory facilities and testbed equipment throughout this project. The publicly available BoT-IoT and CICIoT2023 datasets, and the teams that assembled them, made the offline evaluation component of this work possible.

REFERENCES

[1] C. K. Ejeofobiri, O. O. Victor-Igun, and C. Okoye, "AI-Driven Secure Intrusion Detection for Internet of Things (IoT) Networks," Asian Journal of Mathematics and Computer Research, vol. 31, no. 4, pp. 40–55, 2024.
[2] D. D. Rao, A. A. Waoo, M. P. Singh, P. K. Pareek, S. Kamal, and S. V. Pandit, "Strategizing IoT Network Layer Security Through Advanced Intrusion Detection Systems and AI-Driven Threat Analysis," Journal of Intelligent Systems and Internet of Things, vol. 12, no. 2, pp. 195–207, 2024.
[3] M. Nisha and G. Udhayashri, "AI-Powered Intrusion Detection System for IoT Security," International Journal on Science and Technology (IJSAT), 2024.
[4] S. Kakolu, M. A. Faheem, and M. Aslam, "AI-enabled intrusion detection systems in IoT networks: Advancing defense mechanisms for resource-constrained devices," International Journal of Science and Research Archive, vol. 9, no. 1, pp. 752–769, 2023.
[5] S. A. Bakhsh, M. A. Khan, F. Ahmed, M. S. Alshehri, H. U. Shah, and I. Ullah, "Enhancing IoT network security through deep learning-powered Intrusion Detection System," Internet of Things, vol. 24, p. 100936, 2023.
[6] M. Srinivasan and N. C. Senthilkumar, "Intrusion Detection and Prevention System (IDPS) Model for IIoT Environments Using Hybridized Framework," IEEE Access, 2025.

[7] V. Kantharaju, T. Bheema Lingaiah, K. Nagalakshmi, K. S. Gurumurthy, and S. Chandra, "Machine learning based intrusion detection framework for detecting security attacks in internet of things," Scientific Reports, vol. 14, p. 30275, 2024.
[8] Z. Chiba, N. Abghour, K. Moussaid, O. Lifandali, and R. Kinta, "A Deep Study of Novel Intrusion Detection Systems and Intrusion Prevention Systems for Internet of Things Networks," Procedia Computer Science, vol. 210, pp. 94–103, 2022.
[9] Y. K. Saheed, M. O. Arowolo, A. Balogun, A. Gbolagade, and O. A. Olatunde, "A machine learning-based intrusion detection for detecting internet of things network attacks," Alexandria Engineering Journal, vol. 61, no. 12, pp. 9395–9409, 2022.
[10] R. Kumar, P. Kumar, R. Tripathi, G. P. Gupta, S. Garg, and M. M. Hassan, "A distributed intrusion detection system to detect DDoS attacks in blockchain-enabled IoT network," Journal of Parallel and Distributed Computing, vol. 164, pp. 55–68, 2022.

©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)