FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 10 No. 8, April, 2026, pp 13 – 18
DOI: https://doi.org/10.33003/fjs-2026-1008-4882

BENCHMARKING LIGHTWEIGHT ML MODELS FOR EDGE-IOT BOTNET DETECTION UNDER RESOURCE CONSTRAINTS

***Haruna Aliyu, Yusuf Salisu Ibrahim and Aliyu Suleiman Muhammed**

Department of Engineering, Faculty of Computing, Nile University of Nigeria, Abuja-FCT, Nigeria.

*Corresponding authors' email: 246370036@nileuniversity.edu.ng

ABSTRACT

Increased use of Internet of Things (IoT) devices has resulted in the introduction of cyber vulnerabilities, necessitating automated Intrusion Detection Systems (IDS) at the network edge, where the decisions are localized. A significant research gap exists because proposed lightweight machine learning models are often evaluated on high-performance workstations, misrepresenting actual edge gateway resource constraints. This paper describes in detail the detection of IoT botnets experimentally under simulated hardware limitations using the three ensemble models: XGBoost, LightGBM, and Deep Forest. A Docker-based simulation environment was limited to 1 vCPU and 512 MB RAM. The methodology incorporated stratified sampling of the Bot-IoT dataset, preprocessed via SMOTE oversampling and Random Forest Feature Importance (RFFI). Alongside standard classification metrics, practical resource metrics such as inference latency, memory footprint, throughput, and energy consumption were measured. Achieving a zero False Positive Rate, XGBoost came out with the shortest inference latency (0.41 ms), highest throughput (2, 417 packets/sec), and lowest energy consumption (2.05 J), thus identifying it as the best fit for online, battery-powered deployments. On the other hand, LightGBM had the least memory usage (152 MB), so it can be considered for very RAM-constrained legacy devices. Deep Forest recorded a very high accuracy but used 25 times more energy than XGBoost, thus ruling it out for extreme edge applications. All models exceeded 99.98% accuracy. This paper presents a containerized and reproducible benchmarking framework to measure the energy efficiency, memory, and latency which offers scientifically grounded guidelines for lightweight IDS deployment in the Edge-IoT environments.

Keywords: Edge-IoT; Botnet Detection; Intrusion Detection System; XGBoost; LightGBM; Deep Forest; Lightweight Machine Learning

INTRODUCTION

The rapid growth of the Internet of Things (IoT) has completely transformed the digital landscape. According to market reports, there were approximately 18.5 billion active IoT devices in 2024 and this figure is projected to reach 21.1 billion by the end of 2025 (IoT Analytics, 2025). Edge Computing is being recognized as one of the most effective ways to handle the massive data that these devices generate locally so as to avoid the high latency and overdependence on centralized cloud systems. However, this decentralization has exposed edge nodes to a high risk of cyber attacks. Hackers exploit the typical vulnerabilities in IoT devices, such as the use of default passwords and outdated firmware, to take control of these devices and create botnets that can execute large scale Distributed Denial of Service (DDoS) attacks. The Mirai botnet is one striking example of this risk as it used hundreds of thousands of compromised devices to conduct attacks of over 1 Terabit per second in 2016 (Marzano et al., 2018). The new forms of this type of threat include Aisuru (2024) and BadBox 2.0 (2025) which show continuing changes in the threat landscape (Krebs, 2025; Asimily, 2025). Technology like Machine Learning (ML) and Deep Learning (DL) has played a major role in implementing efficient on-device Intrusion Detection Systems (IDS). However, a major limitation of resources leaves only a few options for deployment: Edge gateways work with very tight constraints concerning processing capability, RAM, and power. Deep neural networks which require hundreds of megabytes of memory and billions of floating-point operations (FLOPs) are simply out of the question for devices such as Raspberry Pi 3 Model A+ (512 MB RAM) or microcontrollers like the ESP32 (512 KB SRAM) (Lundqvist et al., 2025). The usage of very resource-demanding models on these devices results in instability, extraordinary latency, and quick battery drain, thereby ruining their operational value (Jamshidi et al., 2025). During the 5G period, sending all telemetry back to the cloud also results in latency that is a breach of the real-time demands of mission-critical applications. This makes decentralized, edge-native security detection an indispensable feature (Jamshidi et al., 2025).

A number of studies have suggested lightweight ML options. For example, one of them published in the FUDMA Journal of Sciences (FJS) last year showed that hybrid ensemble machine learning models could perform well and be quite robust when classifying intrusions in the IoT environment. Besides, they made very compelling arguments as to why finding a balanced point for achieving a high detection accuracy level and low computational cost is of utmost importance (Eguavoen et al., 2025). In the domain of structured data coming from analysis of network logs, gradient boosting algorithms, in general, play a very big role: XGBoost is not only built on its use of L1 and L2 regularization to address overfitting problems specifically in the case of IoT traffic anomalies, but it has also become the leading model in the field (Belachew et al., 2025; Chen & Guestrin, 2016). LightGBM is a Microsoft product that uses techniques: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) to train theoretically faster and consume less memory (Ke et al., 2017; Rupanetti & Kaabouch, 2025). Deep Forest (gcForest) acts as a non-neural alternative that imitates a hierarchical feature abstraction in a cascaded manner of Random Forest layers without the need for a GPU or backpropagation (Zhou & Feng, 2019). Tree-based explainable approaches such as ELAI (Explainable and Lightweight AI) have also gained prominence because

transparency in decision-making is operationally critical for security operators (Rahmati, 2025).

Nonetheless, major issues remain unresolved after these proposals. For example, most comparative analyses such as Rupanetti and Kaabouch (2025) and Hejazi et al. (2025) test the lightweight claims only on powerful computers with large RAM (16 GB+), thus not reflecting the challenging realities of physical edge devices. Besides, energy consumption and memory usage which are the main Green IoT metrics by Jamshidi et al. (2025) through 'energycomplexity coupling' are rarely taken into account along with accuracy checkpoints. Most importantly, there hasn't been any study that has done a direct side-by-side comparison of XGBoost, LightGBM, and Deep Forest running on containerized environments that have very strict resource limitations (512 MB RAM, 1 vCPU). So, the issue of which model offers the best Edge-IoT trade-off in the real world situation is still an open question.

This study contributes to the literature by exposing through detailed, fully reproducible experiments the performance of three machine learning algorithms, XGBoost, LightGBM, and Deep Forest when used for IoT botnet detection in a virtual environment with limited resources. The objectives are to: (i) design a Docker-based benchmarking framework that can be reliably reproduced and that limits the use of resources to 1 vCPU and 512 MB RAM; (ii) implement and enhance the performance of the three models through a uniform data preprocessing method; (iii) evaluate classification performance by using different metrics such as accuracy, precision, recall, F1-Score and False Positive Rate; and (iv) measure the real computational cost in regard to memory footprint, inference latency, throughput, and energy consumption in order to select the best model for deployment. The article is the first to show a direct, containerized, and multi-metric comparison of these three model architectures, which not only fills the gap of data-backed deployment guidelines in the literature but also assists in better understanding of the model selection for Green IoT security.

MATERIALS AND METHODS

Dataset

The primary benchmark used was the Bot-IoT dataset from the UNSW Canberra Cyber Range Lab (Moustafa, 2021). This dataset has about 3, 668, 522 records and 19 columns. These features include 10 pre-selected network flow features statistically, which were obtained through Correlation and Information Gain analysis from 46+ original attributes; flow identifiers and class labels are also included. The attack categories are DDoS (TCP, UDP, HTTP), DoS, OS Fingerprinting, and Data Exfiltration. Besides, IoT-specific protocols like MQTT and CoAP are also covered. The size of the pre-selected features file is 350-400 MB in CSV format. Major flow features included sequence number (seq), standard deviation (stddev), inbound connections per source IP (N_IN_Conn_P_SrcIP), flow duration statistics (min, mean, max), protocol state (state_number), source and destination packet rates (srate, drate), and inbound connections per destination IP (N_IN_Conn_P_DstIP). Target labels consisted of binary classification (0 = Normal, 1 = Attack), attack category (DDoS, DoS, Reconnaissance, Theft), and subcategory (HTTP, TCP, UDP, OS_Fingerprint, Keylogging, Data_Exfiltration).

Data Preprocessing Pipeline

Initially, raw network logs needed thorough preprocessing before being used for training the model. During data cleaning, all missing (NaN) and infinite (Inf) values were discarded. Categorical features (like proto, service, and state) were converted to numerical values using Label Encoding. The use of this method instead of One-Hot Encoding was mainly due to the fact that it has a low memory consumption, i.e., the memory complexity remains constant at O(1) per feature. Min-Max Scaling normalized all the feature values to the same range, namely the interval [0, 1], by performing the operation given in equation 1:

$$X_{norm} = (X - X_{min}) / (X_{max} - X_{min}) \quad (1)$$

In the above, X_{norm} corresponds to the normalized feature value, X is the initial, unprocessed feature value, X_{min} is the lowest value of the considered feature throughout the dataset, and X_{max} is the highest one. This mathematical capping guarantees that all numerical inputs are scaled proportionately without altering relative variance.

Because the dataset had extreme imbalance between classes, a two-stage sampling approach was utilized. Initially, using Stratified Random Sampling, a representative 10% portion of the whole dataset was taken while keeping the same attack-to-normal traffic ratio as that in the original data (Bankó et al., 2025). Next, the Synthetic Minority Oversampling Technique (SMOTE) (Popoola et al., 2021) was used only on the training set to change the class distribution to a 50:50 ratio. SMOTE creates synthetic instances of the minority class by performing an interpolation between nearest neighbors in the feature space. The test set was completely untouched, thus preserving its original imbalance so that the evaluation metrics would rightly demonstrate the model's performance in the presence of real-world, skewed traffic situations. This way also ensures that there is no data leakage and that accuracy scores are not artificially high (Ali et al., 2025).

Feature Selection

Dimensionality was reduced through the application of RFFI based on MDI, as edge devices will not be able to handle hundreds of redundant features efficiently. The importance of feature j was determined as in equation 2:

$$\text{Importance}_j = \Sigma_t \in \text{Nodes}_j \, \Delta I(t) \quad (2)$$

Where Importance_j is the calculated overall importance of feature j, Nodes_j represents the set of all decision tree nodes across the entire random forest where feature j is actively used to split the data, and $\Delta I(t)$ denotes the measured decrease in impurity (such as Gini impurity or Information Gain) achieved by the data split at node t.

The most important features (N = 15) that together explained 95% of the predictive power were chosen, satisfying the condition: $\Sigma_{i=1}^{R} I_i \geq 0.95 \times I_{total}$. The primary features identified by the algorithmic extraction are detailed in Table 1.

Table 1

Rank	Feature Name	Importance Score	Description of Feature
1	flow_duration	0.284	Total duration of the network flow
2	fwd_packet_length_std	0.192	Standard deviation of forward packet lengths
3	proto	0.145	Textual representation of the network protocol
4	state_number	0.082	Numerical representation of the protocol state
5	seq	0.061	Sequence number of the network flow

Rank	Feature Name	Importance Score	Description of Feature
6	srate	0.054	Source-to-destination packets per second
7	drate	0.048	Destination-to-source packets per second
8	max	0.035	Maximum duration of active flows
9	mean	0.029	Mean duration of active flows
10	N_IN_Conn_P_SrcIP	0.02	Inbound connections per source IP

Equally, the main features identified by the team were flow_duration, fwd_packet_length_std and proto, which are in line with known botnet traffic characteristics (Singh et al., 2024). The use of dimensionality reduction in this case had a major impact on reducing inference latency by drastically cutting the size of the model's input vector, an advantage also pointed out by Sharma et al. (2025).

Model Implementation and Configuration

Three different models were built and tested. One model corresponds to a specific method of ensemble learning:

XGBoost (Extreme Gradient Boosting) was the first model used and it is a very popular one in the industry. Its target function has two components viz. a loss term and a regularization part:

$Obj(\Theta) = L(\theta) + \Omega(\Theta)$

Where $Obj(\Theta$ is the objective function being optimized, $L(\theta)$ represents the differentiable convex loss function measuring the difference between predicted and actual targets, and $\Omega(\Theta)$ is the penalty term enforcing L1 (Lasso) and L2 (Ridge) regularization. This form of regularization helps to avoid overfitting the noise, which is typically high, in the sparse IoT traffic data (Chen & Guestrin, 2016). XGBoost applies a tree growth strategy that is level-wise.

LightGBM (Light Gradient Boosting Machine) is a tool developed by Microsoft that builds tree leaf-wise and features two major optimizations (Ke et al., 2017): (i) GOSS (Gradient-based One-Side Sampling) keeps the instances with large gradients, and those with small gradients are randomly sampled. This reduces the effective size of the dataset, but the accuracy is still preserved; and (ii) EFB (Exclusive Feature Bundling) is a feature dimensionality reduction technique that operates by grouping mutually exclusive features. The formula for calculating LightGBM split gain is:

$Gain = 0.5 \times [GL^2/(HL + \lambda) + GR^2/(HR + \lambda) - (GL + GR)^2/(HL + HR + \lambda)]$

Where Gain represents the calculated improvement from splitting a leaf, G_L and G_R stand for the sums of first-order gradients for the left and right child nodes respectively, H_L and H_R represent the sums of second-order gradients (Hessian) for the left and right child nodes, and λ is the L2 regularization parameter.

The idea to use Deep Forest (gcForest) was to have a non-neural method to deep learning. This approach is based on a succession of Random Forest and Completely Random Forest layers. From each layer, class-probability vectors are produced. These vectors along with the original feature vectors are input to the next layer. Through this, the hierarchical feature abstraction of deep neural networks is imitated without the necessity of backpropagation, matrix multiplications, or GPU usage (Zhou & Feng, 2019).

Since tree-based ensemble models rely on mathematical splitting criteria (such as Information Gain) rather than neural network activation functions, hyperparameter optimization focused exclusively on structural attributes. XGBoost's grid search space included learning_rate [0.01, 0.1, 0.2], max_depth [3, 5, 7], and n_estimators [100, 200, 300]. LightGBM evaluated num_leaves [31, 50, 100] and learning_rate [0.01, 0.05, 0.1]. Deep Forest was evaluated across [2, 4, 6] cascade layers with 100 trees per internal forest.

RandomizedSearchCV with 5-fold cross-validation was used for hyperparameter tuning of the three models. . A 5-fold cross-validation dynamically partitions the training dataset into 5 equal-sized folds (each fold containing exactly 20% of the training data); the model iteratively trains on 4 folds (80%) and validates on the remaining 1 fold (20%). Major parameters were n_estimators (trees or boosting rounds), max_depth (trees depth control), learning_rate (step size for each boosting round), and num_leaves (only for LightGBM, controlling leaf-wise growth complexity). Randomized search was chosen over the grid search mainly due to the computational efficiency and the ability to find near-optimal configurations.

Docker-Based Edge Simulation

To emulate physical edge hardware like Raspberry Pi 3 Model A+ and Raspberry Pi Zero WH, we restricted the inference evaluation to run in a Docker container with the resource limits being strictly enforced by Linux cgroups: --cpus="1.0" (single CPU core) and -memory="512m" (512 MB RAM). The host system was a macOS or Windows 11 equipped with 16 GB RAM and 8 CPU cores; however, models' evaluations were never performed directly on the host. Such a containerized strategy ensures hardware-agnostic reproducible results, and in case the containerized model doesn't work or has a high latency, it is then considered an undeniable demonstration of the model being physically unsuitable for edge deployment regardless of its accuracy. These steps are very much in line with the "practicality void" identified in the literature review (Kallimani et al., 2023; Mostafavi, 2025) as the major challenge.

Performance Evaluation Metrics

The evaluation framework was multifaceted; it measured security and sustainable Green IoT performance simultaneously. From the confusion matrix the following classification metrics were obtained: Accuracy, Precision, Recall, F1-Score (harmonic mean of Precision and Recall, which was chosen as the primary metric because of class imbalance), and False Positive Rate (FPR). Real resource metrics were taken with docker stats at 1 Hz frequency during inference: Inference Latency (average time in ms to classify one network flow sample); Throughput (network flows classified per second); Peak Memory Footprint (maximum RAM consumed in MB during the inference phase); and Energy Consumption Proxy (Joules), which is based on integrating the CPU utilization percentage over inference time this method was adapted from Jamshidi et al. (2025) in order to apply the 'energy-complexity coupling' idea of batterypowered IoT devices that.

RESULTS AND DISCUSSION

Data Preprocessing Outcomes

Initial data analysis revealed a very high-class imbalance in the original Bot-IoT dataset, with the amount of malicious traffic (DDoS/DoS) being far more than benign traffic. Synthetic Minority Over-sampling Technique (SMOTE) was

used only for the training set, so a balanced 50:50 class distribution was reached without the test set getting contaminated. Random Forest Feature Importance (RFFI) reduced the number of features from more than 40 original ones to the top 20 most discriminating ones. Top three features, flow_duration, fwd_packet_length_std, and proto, match botnet behavioural signatures that are widely known: For example, when botnets like Mirai produce traffic at fixed intervals with uniform packet sizes, packet length standard deviation is very low; on the other hand, human-generated legitimate traffic is characterized by high temporal and size variance (Singh et al., 2024). This reduction in the number of dimensions greatly helped us to produce the low latency results from Phase 2.

Baseline vs. Fine-Tuned Model Performance Improvements To quantify the impact of hyperparameter optimization, the models were initially evaluated using their default baseline parameters prior to applying the tuned settings derived from the 5-fold cross-validation process. Table 2 details the comparative improvements achieved across all classification metrics, including the ROC-AUC and PR-AUC.

Table 2: Comparison of Models Before (Baseline) and After Fine-Tuning

Model State	Accuracy	Precision	Recall	F1-Score	FPR	ROC-AUC	PR-AUC
XGBoost (Baseline)	0.9995	0.9997	0.9994	0.9995	0.0012	0.9997	0.9996
XGBoost (Fine-Tuned)	0.9998	0.9999	0.9998	0.9998	0	1	0.9999
LightGBM (Baseline)	0.9994	0.9995	0.9995	0.9995	0.4015	0.9985	0.999
LightGBM (Fine-Tuned)	0.9999	0.9999	0.9999	0.9999	0.3333	0.9998	0.9998
Deep Forest (Baseline)	0.9991	0.9992	0.9993	0.9992	0.4501	0.998	0.9985
Deep Forest (Fine-Tuned)	0.9999	0.9999	0.9999	0.9999	0.3333	0.9999	0.9998

The data in Table 2 unequivocally reveal the significance of fine-tuning. Whereas baseline models were capable of attaining a very high accuracy almost equal to 99.93% on average, hyperparameter tuning produced better and more stable results. Most importantly, fine-tuning XGBoost was so effective that it capped the number of false positives to zero, thus practically eliminating false positives by decreasing FPR from 0.0012 to a perfect 0.0000. Moreover, fine-tuning improved the ROC-AUC and PR-AUC scores for all models to the highest possible levels, demonstrating exceptional performance in class separation regardless of the threshold used.

Standard Classification Performance

All three fine-tuned models performed quite beautifully being almost perfect classifying the held-out, imbalanced test set, as depicted in Table 2. Yet, very crucial dissimilarity was observed when the False Positive Rate (FPR) was analyzed. Only XGBoost disclosed a FPR of 0.0000 whereas LightGBM and Deep Forest both shared the same FPR of 0.3333.

FPR difference is indeed a very big factor in operations when looked from the perspective of network security. A FPR of 0.3333 means one third of benign traffic flows are misclassified as threats, which not only creates so much 'alert fatigue' among network administrators but also could impact the automated mitigation systems that may block legitimate traffic. The ability of XGBoost to perfectly bifurcate benign and attack traffics gives it an edge. Besides, even with a slightly lower nominal accuracy (0.9998 Vs. 0.9999), it is this feature that sets it apart for automated, unsupervised intrusion blocking in production environments. In fact, this discovery is at odds with the common belief that Deep Learning-inspired architectures (like Deep Forest) inevitably lead to more trustworthy classifications (Hejazi et al., 2025).

Practical Resource Performance

Performance under the simulated Docker edge environment (1 vCPU, 512 MB RAM) is presented in Table 3. The results reveal stark differences in hardware efficiency that are invisible to standard accuracy-based comparisons.

Table 3: Practical Hardware Performance Metrics under Docker Edge Simulation (1 Vcpu, 512 MB RAM)

Model	Inference Latency (ms)	Throughput (Packets/sec)	Memory Footprint (MB)	Energy Consumption (J)
XGBoost	0.4137	2,417.23	168.30	2.05
LightGBM	1.4637	683.20	152.12	7.20
Deep Forest	10.6049	94.30	158.20	53.05

Inference Latency and Throughput: XGBoost had the lowest inference latency (0.4137 ms per sample) and the best throughput (about 2,417 packets per second), thus easily supporting real-time traffic analysis. LightGBM was around 3.5 times slower (1.4637 ms) which goes against the general notion that LightGBM is faster than XGBoost at runtime. This is important: the leaf-wise growth of LightGBM gives it an advantage mainly at the training stage; during inference row-by-row in a containerised, single-core scenario, it is XGBoost's inference engine that seems better optimised. Deep Forest was really slower, at 10.60 ms, around 25 times the latency of XGBoost, due to the computationally expensive cascade forest layers that need sequential multi-layer processing for each inference call.

Memory Footprint: Among all, LightGBM used the minimum peak RAM of 152.12 MB only, which confirmed its memory efficiency superiority theoretically with GOSS and EFB implementations (Ke et al., 2017; Rupanetti & Kaabouch, 2025). In contrast, unexpectedly, XGBoost led the memory consumption list at 168.30 MB, which was nearly 16 MB more than that of LightGBM, and Deep Forest took 158.20 MB that put it into the middle of the memory efficiency ordering. Nevertheless, all three models could be said to run very well within the 512 MB limitation by absolute figures.

Energy Consumption: Compared to its peers, XGBoost's energy consumption was the minimal, just 2.05 Joules. LightGBM used 7.20 Joules (about 3.5 times more) mainly due to the extended duration of its inference time during which the CPU had to be engaged continuously. Deep Forest, however, drank in 53.05 Joules, roughly 25 times more than XGBoost. This verified the 'energy-complexity coupling' that Jamshidi et al. (2025) claimed: the complexity of layers in the cascade forest algorithm led to direct energy/battery draining. Models which finish inference faster give the processor a

chance to go idle state earlier thus offering a significantly lower total energy consumption a very important indicator for battery-operated IoT sensors.

Trade-off Analysis and Deployment Scenario Assessment
Since all three models achieved classification accuracy exceeding 99.98%, deployment decisions must be determined by hardware efficiency metrics. Table 4 presents a normalised trade-off matrix comparing the three models relative to the best performer in each dimension.

Table 4: Operational Trade-Off Matrix (Normalized To Best Performer per Metric)

Model	Relative Speed	Relative Memory Cost	Relative Energy Cost	Optimal Deployment Tier
XGBoost	1.0× (Baseline)	1.10×	1.0× (Baseline)	Industrial Gateway / Battery-Critical Sensor
LightGBM	3.53× Slower	1.0× (Baseline)	3.51× Higher	Legacy Sensors (RAM-Limited, <160 MB)
Deep Forest	25.63× Slower	1.04×	25.87× Higher	High-Resource Edge (Non-Time-Critical Only)

Three usage deployment scenarios were selected to evaluate based on these results. For Scenario A, industrial IoT gateways (that are mains-powered, high traffic), XGBoost is quite evidently the best choice, managing to process more than 2,400 packets per second while also having a latency of 0.41 ms and zero false positives, hence industrial operations will not be interrupted by false alarms. Scenario B, Remote sensors that are battery-powered XGBoost also outperforms other models: in fact, its energy consumption (2.05 J) is less than one-third of LightGBM's (7.20 J), which means the battery life of a device can be significantly increased. This challenges the traditional belief that LightGBM is the best choice for devices with a battery constraint (Rupanetti & Kaabouch, 2025). Scenario C, legacy hardware with a tightly limited RAM (below approximately 160 MB) LightGBM is the more reasonable choice with a footprint of 152.12 MB, therefore allowing for less chance of the system running out of memory and crashing. Deep Forest should not be used for real-time edge security/detection purposes because of both its high latency (10.60 ms) and energy consumption (53.05 J), despite its classification accuracy being rather competitive.

Comparison with Previous Research and System Impact to present this research among the global academic conversation, it is logical to highlight the differences of these results with the recent studies. According to Hejazi et al. (2025) and Rupanetti and Kaabouch (2025), Deep Learning and LightGBM models are excellent choices. Their systems, however, were tested mainly on workstations with plenty of resources. Our resource-limited experiments not only support Rupanetti and Kaabouch's points about LightGBM being more memory efficient but LightGBM even managed to run within the 160 MB memory limit. Our results also significantly dispute the majority opinion that LightGBM is fastest especially when running on a single CPU core.One can say that in this limited setup, the running time of XGBoost was 3.5 times shorter than that of LightGBM in a sequential inference scenario.

Moreover, the huge difference in energy consumption of XGBoost (2.05 J) and Deep Forest (53.05 J), which is almost 26 times, strongly supports "energy-complexity coupling" paradigm put forward by Jamshidi et al. (2025). In fact, it clearly shows that complex model architectures cause heavy energy consumption on edge devices, a problem that is completely hidden in assessments that focus on accuracy only and not system efficiency. Ultimately, these results prove that when applied to strict resource-constrained systems, models must be explicitly selected based on containerized performance profiles rather than generalized theoretical assumptions.

CONCLUSION

This research showed that selecting a machine learning model for Edge-IoT intrusion detection solely based on accuracy is not enough. Performance testing of XGBoost, LightGBM, and Deep Forest in a Docker-simulated edge environment (1 vCPU, 512 MB RAM) with the BotIoT dataset revealed several key operation differences which were not obvious from the fact that the three models had almost identical F1-scores.

XGBoost turned out to be the best option in most edge deployment situations as it had the lowest inference latency (0.41 ms), highest throughput (2, 417 packets/sec), lowest energy consumption (2.05 J), and, most importantly, a zero False Positive Rate. For gateways that are powered by the mains and battery-sufficient sensors, XGBoost is the recommended model whenever RAM capacity is above 170 MB. LightGBM is still the right choice only if the RAM is extremely limited (below around 160 MB), where its 16 MB memory saving over XGBoost can compensate for the slower speed and higher energy consumption. Even though Deep Forest matched the accuracy of the other models, it required highly disproportionate resource costs, consuming 25 times more energy and having 25 times the latency of XGBoost, making it unfit for real-time edge security at this hardware tier.

The containerized benchmarking method used in this study sets a reproducible scenario for lightweight IDS research and connects the gap between theoretical machine learning papers and practical hardware deployment. The main three contributions of this research are: (i) first timed, containerized comparison of XGBoost, LightGBM, and Deep Forest under strict edge resource limitations, thus moving the community away from theoretical F1-score only; (ii) experimental support of the 'energy-complexity coupling' idea, which demonstrated that XGBoost is the best Green IoT solution, and questioned the earlier belief that LightGBM is better for battery-powered equipment; and (iii) a Docker-based evaluation framework that is standardized and can be replicated without special hardware testbeds. The next steps could involve using actual hardware such as Raspberry Pi 3 Model A+ or ESP32 for validation, perform tests on adversarial robustness against intentionally created evasion traffic, and include Federated Learning frameworks for privacy-preserving distributed edge detection techniques (Roy et al., 2024; Reis, 2025).

REFERENCES

Adeyemi T. S., & Muhammed A. (2024). Botnet attack detection in IoT using machine learning models. Int. J. Sci. Res. Arch., **12**(1), 2221-2229. https://doi.org/10.30574/ijsra.2024.12.1.0936

Ali Z., Tiberti W., Marotta A., & Cassioli D. (2025). Deep Transfer Learning for Intrusion Detection in Edge Computing Scenarios. IEEE Internet of Things J., **12**(21), 44882.

Asimily. (2025). The Top Internet of Things (IoT) Cybersecurity Breaches in 2025. [Online]. https://asimily.com/blog

Bankó M. B., Dyszewski S., Králová M., Limpek M. B., Papaioannou M., Choudhary G., & Dragoni N. (2025). Advancements in Machine Learning-Based Intrusion Detection in IoT: Research Trends and Challenges. Algorithms, **18**(4), 209.

Belachew H. M., Beyene M. Y., Desta A. B., Alemu B. T., Musa S. S., & Muhammed A. J. (2025). Design a Robust DDoS Attack Detection and Mitigation Scheme in SDN-EdgeIoT by Leveraging Machine Learning. IEEE Access.

Chen T., & Guestrin C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785-794. https://doi.org/10.1145/2939672.2939785

Eguavoen, V. O., Olanrewaju, B. S., & Okafor, C. N. (2025). A hybrid CNN-LSTM and AdaBoost model for classifying intrusion in IoT networks. FUDMA Journal of Sciences (FJS), 9(5), 204-212. https://doi.org/10.33003/fis-2025-0905-3495

Hejazi S. M., Alshalabi A. Y., Hatamleh M., & Albaroudi E. (2025). A Lightweight Hybrid Deep Learning-Based Intrusion Detection System for Detecting Botnet Attacks in IoT

Networks. J. Sci. Res. Rep., **31**(11), 97-120. https://doi.org/10.9734/jsrr/2025/v311113654

IoT Analytics. (2025). State of IoT 2025: Number of connected IoT devices growing 14% to 21.1 billion. [Online]. https://iot-analytics.com

Jamshidi S., Erfan F., Abdul-Wahab O., Bellaiche M., & Khomh F. (2025). Lightweight Autoencoder-Isolation Forest Anomaly Detection for Green IoT Edge Gateways. arXiv preprint arXiv:2511.18235.

Kallimani R., Pai K., Raghuwanshi P., Iyer S., & López O. L. A. (2023). TinyML: Tools, Applications, Challenges, and Future Research Directions. arXiv preprint arXiv:2303.13569.

Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., & Liu T. Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. Adv. Neural Inf. Process. Syst., **30**.

Krebs B. (2025). DDoS Botnet 'Aisuru' Blankets U.S. ISPs in Record DDoS. KrebsOnSecurity. [Online]. https://krebsonsecurity.com

Lundqvist T., et al. (2025). Lightweight Machine Learning Models for Intrusion Detection on IoT Devices. ResearchGate.

Marzano A., Alexander D., Fonseca O., Fazzion E., Hoepers C., Steding-Jessen K., & Meira W. Jr. (2018). The Evolution of Bashlite and Mirai IoT Botnets. 2018 IEEE Symposium on Computers and Communications (ISCC), pp. 813-818.

Mostafavi A. (2025). Lightweight AI Models for IDS in IIoT: Balancing Accuracy and Computational Efficiency. M.Sc. Thesis, Mälardalen University, Sweden.

Moustafa N. (2021). ToN_IoT Datasets. University of New South Wales (UNSW). [Online]. https://research.unsw.edu.au/projects/toniot-datasets

Popoola S. I., Adebisi B., Ande R., Hammoudeh M., Anoh K., & Atayero A. A. (2021). SMOTE-DRNN: A Deep Learning Algorithm for Botnet Detection in the Internet-ofThings Networks. Sensors, **21**(9), 2985.

Rahmati M. (2025). Towards Explainable and Lightweight AI for Real-Time Cyber Threat Hunting in Edge Networks. arXiv preprint arXiv:2504.16118.

Reis M. J. C. S. (2025). Edge-FLGuard+: A Federated and Lightweight Anomaly Detection Framework for Securing 5G-Enabled IoT in Smart Homes. Future Internet, **17**, 179.

Roy S., Li J., & Bai Y. (2024). Federated Learning-Based Intrusion Detection System for IoT Environments with Locally Adapted Model. 2024 IEEE International Conference on Computing, Networking and Communications (ICNC). https://doi.org/10.1109/ICNC57223.2024.10417048

Rupanetti D., & Kaabouch N. (2025). Leveraging Machine Learning for Botnet Attack Detection in Edge-Computing Assisted IoT Networks. arXiv preprint arXiv:2508.01542.

Sharma A., Rani S., & Shabaz M. (2025). An optimized stacking-based TinyML model for attack detection in IoT networks. PLOS ONE, **20**(8), e0329227. https://doi.org/10.1371/journal.pone.0329227

Singh N. J., Hoque N., Singh K. R., & Bhattacharyya D. K. (2024). Botnet-based IoT network traffic analysis using deep learning. Security and Privacy, **7**(2), e355. https://doi.org/10.1002/spy2.355

Zhou Z. H., & Feng J. (2019). Deep Forest. Natl. Sci. Rev., **6**(1), 74-86. https://doi.org/10.1093/nsr/nwy108