Open Access and Peer Reviewed Journal ISSN 2394-2231 https://www.ijctjournal.org/

RESEARCH ARTICLE OPEN ACCESS

Lightweight CNN–BiLSTM Intrusion Detection on the CICIoT2023 Benchmark: Balancing Multi-Class Accuracy and Edge Deployability

Mothanna Abu Judeh[1],*, Adnan H. Al-Helali[2]

[1] *Department of Cybersecurity, Faculty of Information Technology, Irbid National University, Irbid, Jordan*
[2] *Faculty of Information Technology, Irbid National University, Irbid, Jordan*
Corresponding author email: 202511441@inu.edu.jo

Abstract — IoT devices are now everywhere: smart homes, hospitals, factories, and surveillance networks. Most of them ship with weak default security and rarely get patched, which has made them an easy target. Signature-based intrusion detection systems miss attacks they have never seen before, and most deep learning alternatives are too heavy to run anywhere near the devices they are supposed to protect. This paper presents a hybrid model that pairs a one-dimensional convolutional neural network (1D-CNN) with a bidirectional long short-term memory network (BiLSTM). The CNN extracts spatial features from traffic records, and the BiLSTM captures how those features change over time. We train and evaluate the model on CICIoT2023, a recent benchmark containing 33 attacks in seven categories, including distributed denial of service (DDoS), Mirai botnet traffic, reconnaissance, spoofing, and web attacks. Preprocessing consists of class balancing, feature normalization, and feature reduction to keep the model small. The evaluation reports detection quality and computational cost together, comparing the hybrid against classical machine learning baselines and against standalone CNN and BiLSTM ablations, and measuring parameter count and central processing unit (CPU) inference latency rather than accuracy alone. We close by discussing the limits of the current design and two directions we plan to explore next: federated training across devices and explainable outputs for security analysts.

Keywords — Internet of Things; intrusion detection; deep learning; CNN–BiLSTM; CICIoT2023; edge computing.

I. INTRODUCTION

A. Background

The Internet of Things stopped being a niche technology some time ago. IoT Analytics [1] counted around 21.1 billion connected devices by the end of 2025 and expects roughly 39 billion by 2030. These are not laptops or phones. They are cameras, smart meters, medical monitors, badge readers, and industrial sensors: devices that are cheap to buy, easy to install, and very easy to forget about. Many ship with default passwords, expose management ports to the network, and never receive a firmware update after the day they are mounted on a wall.

Attackers noticed early. The Mirai botnet of 2016 remains the clearest example: it scanned the internet for IP cameras and DVRs that still used factory credentials, infected around 600,000 of them, and used the resulting botnet to launch some of the largest DDoS attacks recorded at the time [2]. Nearly a decade later, Mirai variants are still active, which says something uncomfortable about how slowly the IoT ecosystem fixes its problems.

Intrusion detection systems are one of the few defenses that do not require modifying the devices themselves. A signature-based IDS matches traffic against known attack patterns, so it fails on anything it has not seen before. Anomaly-based detection with classical machine learning does better, but it depends on hand-crafted features and tends to degrade as traffic changes. Deep learning removes the manual feature engineering step because the network learns its own representations from raw or lightly processed traffic. The problem for a long time was evaluation: much of the published work relied on aging datasets such as KDD99 and NSL-KDD, which have little in common with modern IoT traffic. The CICIoT2023 benchmark addressed this by executing 33 attacks across seven categories against a topology of 105 real IoT devices [3], and it has quickly become the standard testbed for this line of research.

Recent results on CICIoT2023 show a trade-off that motivates this paper. Large architectures perform well: Tseng et al. [4] applied seven deep learning models, including a Transformer, and reported strong multi-class accuracy. But models of that size assume hardware that a typical IoT gateway does not have. At the other end, Javed et al. [5] showed that very small models can run directly on smart home devices, at the cost of weaker performance when many attack classes are involved. The middle ground, models small

enough for edge hardware but accurate enough for realistic multi-class detection, is less explored.

B. Research Problem

This paper asks a narrow question: can a compact hybrid model, a one-dimensional CNN feeding a bidirectional LSTM, match the detection performance of heavier architectures on CICIoT2023 while staying small enough to deploy at the network edge? Three sub-questions follow from it. First, how does the hybrid model perform on binary and multi-class classification compared with classical machine learning baselines and with CNN or LSTM models used alone? Second, what does the model cost in parameters and inference time? Third, does combining spatial feature extraction (CNN) with temporal modeling (BiLSTM) actually add anything over either component by itself?

C. Significance of the Study

The practical motivation is direct. Most organizations that run IoT at scale, and this includes nature reserves, hospitals, and municipalities, not just tech companies, place their devices behind gateways with modest hardware. Surveillance cameras and NVRs are a good example: they sit on the network for years, they are rarely patched, and they were the raw material of Mirai. A detection model that runs on the gateway itself can flag an infection before it spreads, without waiting for traffic to reach a central server.

There is also an academic gap. Most published CICIoT2023 studies report accuracy, precision, and recall, and stop there. Far fewer report model size or inference latency, even when the stated goal is IoT deployment. Measuring detection quality and computational cost together gives a more honest picture of whether a model is deployable, and that is the comparison this paper sets out to make.

D. Objectives

The study has four objectives:

1) Build a preprocessing pipeline for CICIoT2023 covering class balancing, feature normalization, and feature reduction.

2) Design and train a compact hybrid model in which a 1D-CNN extracts spatial features from traffic records and a BiLSTM captures temporal dependencies.

3) Evaluate the model with accuracy, precision, recall, and F1-score on both binary and multi-class tasks.

4) Measure parameter count and inference latency, and compare the results against classical baselines and against standalone CNN and BiLSTM models.

E. Organisation of the Paper

The remainder of the paper is organised as follows. Section II reviews related work on deep learning intrusion detection for IoT and positions this study within it. Section III describes the dataset, tools, and methods, states the model formally, and records the experimental setup. Section IV reports the results, discusses them, and compares them with prior work. Section V concludes and points to future directions, including federated training and explainable outputs.

II. RELATED WORK

This section surveys work on machine learning and deep learning intrusion detection for IoT networks. It is organised by approach rather than by date, moving from classical machine learning to single-architecture deep models, then to hybrid CNN–recurrent designs, and finally to the small subset of studies that treat computational cost as a first-class concern. The section closes with a critical comparison and a statement of the gap this paper addresses.

A. Classical Machine Learning Approaches

Early anomaly-based IDS research for IoT relied on classical classifiers: decision trees, Random Forest, support vector machines, and naive Bayes, usually trained on hand-engineered flow statistics. The authors of CICIoT2023 themselves benchmarked several of these methods on their dataset and found that tree-based ensembles performed respectably on binary classification [3]. The appeal is obvious. These models train in minutes, they run on modest hardware, and a decision tree can be inspected by a human analyst.

Their weakness shows up in two places. The first is feature engineering: performance depends heavily on which flow statistics an expert chose to compute, and those choices do not transfer well between networks. The second is multi-class performance. Distinguishing benign traffic from an attack is a much easier problem than distinguishing 33 attack types from each other, and classical models degrade noticeably as the number of classes grows. Because IoT traffic is also severely imbalanced, with DDoS floods producing millions of records while brute-force attempts produce a few thousand, minority classes are frequently absorbed into the majority.

B. Single-Architecture Deep Learning Models

Deep learning removed the manual feature engineering step. Convolutional networks applied to traffic records treat a feature vector as a one-dimensional signal and learn local patterns across adjacent features. Recurrent models, particularly LSTM and its bidirectional variant, model traffic as a sequence and learn dependencies between successive packets or flows. Hizal et al. [6] built a deep learning IDS aimed specifically at IoT DDoS traffic and reported that deep models handled the scale and variety of IoT data better than the classical baselines they compared against.

The most thorough single-architecture comparison on CICIoT2023 comes from Tseng et al. [4], who trained seven deep learning models, including a Transformer, and evaluated them on both binary and multi-class tasks. Their multi-class results were strong, and the paper is valuable precisely because it did not stop at binary classification, which is where much of the earlier literature stopped. Wahab et al. [7] reported a similar pattern on the same dataset: their DNN, CNN, and Transformer models all exceeded 99% on binary classification and fell to roughly 92 to 93% once twelve attack classes were involved. That gap between binary and multi-class accuracy is consistent across the literature and is worth keeping in mind when a paper reports only the binary number.

Neither study reports model size or inference time. A Transformer with multi-head attention over a long feature sequence is not something a typical IoT gateway can run, and the omission matters when the stated application is IoT security.

C. Hybrid CNN–Recurrent Architectures

The intuition behind hybrid models is that convolutional and recurrent layers capture different things. The CNN finds local structure within a record; the recurrent layer finds structure across records. Combining them should, in principle, beat either alone.

Altunay and Albayrak [8] built a CNN+LSTM IDS for industrial IoT networks and found the hybrid outperformed the individual components. Nazir et al. [9] reached the same conclusion with a hybrid CNN–LSTM evaluated across several IoT threat categories. Gueriani et al. [10] applied a CNN–LSTM to CICIoT2023 and validated on CICIDS2017, reporting 98.42% accuracy with an F1-score of 98.57%. Their false positive rate of 9.17% is high enough to be a practical problem: at IoT traffic volumes, roughly one benign flow in eleven raising an alert would bury an analyst, and the paper does not dwell on this.

Bidirectional variants push the idea further. A BiLSTM reads a sequence in both directions, which helps when the signature of an attack only becomes clear from what follows it. Sadhwani et al. [11] designed a BiLSTM–CNN hybrid for IoT and compared it against a long list of prior architectures, including 1D-CNN, two-layer BiLSTM, MSCNN-LSTM, and CNN–LSTM, reporting consistent gains. Sinha et al. [12] reported very high figures for an LSTM–CNN framework on BoT-IoT, close to 99.9% accuracy. Numbers that high on a single dataset invite caution rather than confidence; BoT-IoT is known to be relatively easy to separate, and the result may not survive a harder benchmark.

Two structural questions are unresolved in this literature. The first is ordering: some studies feed the CNN output into the recurrent layer, others reverse it, and few justify the choice with an ablation. The second is class imbalance. Peng et al. [13] addressed it directly with a CNN–BiLSTM trained using focal loss instead of standard cross-entropy, which is one of the more principled treatments of the problem in this space.

D. Lightweight and Edge-Deployable Systems

A smaller group of studies treats computational cost as a design constraint rather than an afterthought, and this is the group closest to the present work.

Javed et al. [5] implemented machine learning IDS directly on smart home IoT devices, demonstrating that on-device detection is feasible on constrained hardware, though their models are simple and their evaluation does not extend to fine-grained multi-class detection. Alzahrani et al. [14] built an energy-aware CNN–LSTM for fog computing environments and, unusually, reported latency, energy consumption, false alarm rate, and detection rate together, comparing CICIoT2023 against KDD99 and NSL-KDD. Their paper is a useful model for how deployability should be reported.

The most directly relevant study is Jouhari and Guizani [15], who proposed a lightweight CNN–BiLSTM specifically for resource-constrained IoT devices. On UNSW-NB15 they reported 97.28% accuracy for binary classification and 96.91% for multi-class, with a small parameter count and low inference latency. Their work establishes that the CNN–BiLSTM combination can be compressed without losing much accuracy. It also defines the boundary of what has already been shown, and therefore what the present study must do differently, a point taken up in Section II-F.

E. Critical Comparison

Table I summarises the studies discussed above along the two dimensions that matter here: detection performance and whether computational cost was measured.

TABLE I. *Comparison of representative deep learning IDS studies for IoT.*

Study	Architecture	Dataset	Reported performance	Cost reported?
Altunay & Albayrak [8]	CNN + LSTM	UNSW-NB15, X-IIoTID	High accuracy on IIoT traffic	No
Gueriani et al. [10]	CNN + LSTM	CICIoT2023, CICIDS2017	98.42% accuracy, F1 98.57%, FPR 9.17% (binary)	No
Nazir et al. [9]	Hybrid CNN–LSTM	Multiple IoT sets	Strong detection across threat types	Partial

Study	Architecture	Dataset	Reported performance	Cost reported?
Tseng et al. [4]	Transformer + 6 DL models	CICIoT2023	Strong binary and multi-class results	No
Jouhari & Guizani [15]	Lightweight CNN–BiLSTM	UNSW-NB15	97.28% binary, 96.91% multi-class	Yes (size, latency)
Alzahrani et al. [14]	Lightweight CNN + LSTM	CICIoT2023, KDD99, NSL-KDD	High detection, low false alarm rate	Yes (latency, energy)
Javed et al. [5]	Lightweight ML on-device	Smart-home IoT traffic	Runs on constrained home hardware	Yes
Wahab et al. [7]	DNN, CNN, Transformer	CIC-IoT2023	99.2% binary; 93.0% on 12-class	No
Sadhwani et al. [11]	BiLSTM → CNN	UNSW-NB15	Outperforms 1D-CNN, BiLSTM, CNN–LSTM baselines	Partial (CPU vs GPU)
This study	**Lightweight 1D-CNN → BiLSTM**	**CICIoT2023**	**Binary and multi-class (Section IV)**	**Yes (parameters, latency)**

Three observations follow. First, hybrid CNN–recurrent models consistently outperform their individual components, so the architectural premise is well supported. Second, reported accuracy is not comparable across rows, because the datasets differ and because binary and multi-class results are often presented side by side as though they were equivalent. Third, and most importantly for this study, the rightmost column is mostly empty. Of the studies surveyed, only Jouhari and Guizani [15], Alzahrani et al. [14], and Javed et al. [5] report what the model costs to run, and only the first two report it in a form that would let a practitioner decide whether deployment is realistic.

A methodological caveat applies to almost every row. Accuracy figures above 99% on network intrusion datasets often reflect imbalance in the data rather than skill in the model, since a classifier that labels everything as the majority class can score well on accuracy alone. Precision, recall, and per-class F1 are the informative metrics, and papers reporting only headline accuracy should be read with that in mind. This is why the evaluation in Section III reports per-class results rather than a single aggregate figure.

F. Research Gap

The literature supports three claims. Hybrid CNN–recurrent architectures detect IoT intrusions better than single architectures. CICIoT2023 is currently the most realistic public benchmark for this task. And lightweight CNN–BiLSTM models can run on constrained hardware, as Jouhari and Guizani [15] demonstrated.

What has not been established is whether that last result carries over to CICIoT2023. The distinction is not cosmetic. UNSW-NB15, the dataset Jouhari and Guizani used, contains nine attack categories captured on a conventional network testbed. CICIoT2023 contains 33 attacks executed by real IoT devices against other real IoT devices across a 105-device topology, with a far more skewed class distribution [3]. A compact model that separates nine classes cleanly may lose ground when asked to separate 33. Meanwhile the studies that do use CICIoT2023 [4], [7], [10] either use architectures too large for edge deployment or do not report what their models cost to run.

This paper addresses that gap by evaluating a lightweight CNN–BiLSTM on CICIoT2023 and reporting detection performance and computational cost together, with ablations against standalone CNN and BiLSTM models to test whether the hybrid structure earns its added complexity.

G. Summary

Research on IoT intrusion detection has moved from classical classifiers through single deep architectures to hybrid designs, with a recent and still small turn toward models that can actually be deployed. The gap this study occupies sits at the intersection of the last two: a hybrid architecture, evaluated on the current benchmark, measured on both accuracy and cost. Section III describes the dataset, tools, and procedures used to do this.

III. METHODOLOGY

This section describes the data, software, and hardware used in the study, states the proposed model formally, defines the evaluation metrics, and records the experimental configuration. The level of detail is intended to let another researcher reproduce the work.

A. Study Design

The study is quantitative and experimental. A proposed model is trained on labelled data and compared against baseline models under identical conditions, with performance differences taken as evidence for or against the architectural choice. The design is comparative in two directions. Externally, the proposed CNN–BiLSTM is compared against classical machine learning baselines (logistic regression, Random Forest) and against results reported in the literature for the same dataset. Internally, it is compared against a standalone 1D-CNN and a standalone BiLSTM, each trained

on the same data with the same procedure. This second comparison is an ablation, and it is what allows the third research question in Section I-B to be answered: if the hybrid does not beat both of its components, the added complexity is not justified.

Two classification tasks are run. The binary task separates benign from malicious traffic. The multi-class task assigns each record to one of the eight top-level categories in Table II. Reporting both is deliberate, since Section II-B showed that binary results alone overstate how well these models perform.

B. Dataset

The study uses CICIoT2023, produced by the Canadian Institute for Cybersecurity at the University of New Brunswick [3]. The dataset was chosen for three reasons discussed in Section II: it was captured on real IoT hardware rather than simulated, its attacks were launched by compromised IoT devices against other IoT devices, and it is recent enough to contain attack behaviour that resembles what is seen on production networks today.

The testbed comprised 105 devices. Sixty-seven of these communicated actively over Wi-Fi and Ethernet, including smart cameras, sensors, home automation controllers, and microcontrollers, while a further 38 Zigbee and Z-Wave endpoints were attached through five hubs. Attacks were generated with standard offensive tooling, including Nmap, Hping3, and Metasploit, so the traffic reflects techniques an actual attacker would use.

The processed data is distributed as 169 CSV files derived from the original packet captures, holding 46,686,579 records in total. Each record carries 46 numeric features and one label. Of these records, 1,098,195 are benign and 45,588,384 are malicious. Labels resolve to 34 classes: benign traffic plus 33 distinct attacks, which the dataset authors group into seven categories. Table II lists these categories.

TABLE II. *Attack categories in CICIoT2023 and their approximate representation.*

Category	Example attacks in the category	Share of records
DDoS	ICMP/UDP/TCP floods, SYN flood, HTTP flood, fragmentation attacks	Dominant — majority of malicious traffic
DoS	Single-source TCP, UDP, SYN and HTTP floods	Large
Mirai	GRE IP flood, GRE Ethernet flood, UDP plain flood	Moderate
Reconnaissance	Host discovery, port scanning, OS fingerprinting, ping sweep	Small
Spoofing	ARP spoofing, DNS spoofing	Small
Web-based	SQL injection, command injection, XSS, uploading attack, browser hijacking	Very small
Brute force	Dictionary brute force against device credentials	Smallest
Benign	**Normal device-to-device and device-to-internet traffic**	**1,098,195 records (≈2.4%)**

The class distribution in Table II is the central difficulty of this dataset. Benign traffic accounts for roughly 2.4% of all records, and the smallest attack categories are smaller still by several orders of magnitude. A model trained on the raw distribution can reach very high accuracy by predicting the dominant DDoS classes and ignoring everything else, which is precisely the artefact noted in Section II-E. The preprocessing described in Section III-C exists mainly to prevent this.

The 46 features are numeric and describe each traffic record along several dimensions: flow timing (flow duration, inter-arrival time, rate), header properties (header length, protocol type, IP version), TCP flag counts (SYN, FIN, RST, PSH, ACK, ECE, CWR), protocol indicators (HTTP, HTTPS, DNS, Telnet, SMTP, SSH, IRC, TCP, UDP, DHCP, ARP, ICMP), and packet size statistics (minimum, maximum, average, standard deviation, total size, variance, magnitude, radius, covariance, weight). No categorical encoding is required, since all features arrive as numeric values.

C. Data Pre-processing

Preprocessing proceeds in six steps.

First, the 169 CSV files are merged into a single dataset. Second, records with missing or infinite values are removed, and duplicate records are dropped; both occur in the raw files and both distort training if left in place.

Third, a stratified subsample is drawn. Training on all 46.7 million records is not necessary for this study and is not feasible on the available hardware; more importantly, it is standard practice in this literature, and Gueriani et al. [10] worked with roughly 1.19 million records from the same dataset for the same reason. This study draws approximately two million records with the original class proportions preserved, so that the subsample remains representative before balancing is applied.

Fourth, class imbalance is addressed. The majority DDoS and DoS classes are randomly undersampled, and the minority classes, chiefly web-based and brute-force attacks, are

oversampled with SMOTE. Balancing is applied only to the training partition. The test set retains the original distribution, because a model evaluated on artificially balanced test data would report performance it will not reproduce on a live network. This is a common and easily missed error in published work.

Fifth, features are normalized. Because the features span very different ranges, from binary protocol flags to byte counts in the millions, each is scaled to zero mean and unit variance using StandardScaler. The scaler is fitted on the training partition only and then applied to the test partition, which prevents information from the test set leaking into training.

Sixth, dimensionality is reduced. Features with near-zero variance are removed, and pairwise Pearson correlation is computed so that one feature from each pair correlated above 0.95 can be dropped. Fewer input features means a smaller first convolutional layer and therefore a smaller model, which serves the deployability objective directly. The final feature count is reported in Section IV.

D. Proposed System Architecture

The model places a one-dimensional convolutional front end ahead of a bidirectional LSTM. The ordering matters and is not arbitrary. The convolutional layers reduce the input to a compact sequence of learned feature maps before the recurrent layer sees it, which keeps the BiLSTM small, and the BiLSTM is the more expensive component per unit. Section II-C noted that few studies justify this ordering; the ablation described in Section III-A tests it directly.

Table III summarises the layer configuration. Filter counts and units were kept deliberately low, since the objective is a model that fits on an edge device rather than one that maximises accuracy at any cost.

TABLE III. *Layer configuration of the proposed CNN–BiLSTM model.*

Layer	Configuration	Purpose
Input	Reshaped feature vector (n features × 1)	Treats the record as a 1-D signal
Conv1D block 1	32 filters, kernel 3, ReLU, batch normalization	Local feature patterns
MaxPooling1D	Pool size 2	Downsampling, fewer parameters
Conv1D block 2	64 filters, kernel 3, ReLU, batch normalization	Higher-level combinations
Dropout	Rate 0.3	Regularization
BiLSTM	64 units (32 per direction)	Temporal dependencies in both directions
Dropout	Rate 0.3	Regularization
Dense	64 units, ReLU	Feature combination
Output	Sigmoid (binary) or softmax (multi-class)	Classification

The formal description of each layer, together with the loss function and optimizer, appears in Sections III-E through III-I.

E. Problem Formulation

Let the preprocessed dataset be a set of labelled records:

$$D = \{(x_i, y_i)\},\ i = 1 \dots N,\ x_i \in R^d \quad (1)$$

where d is the number of features surviving the reduction step in Section III-C and N is the number of records after balancing. For the binary task the label is $y_i \in \{0, 1\}$, with 0 for benign and 1 for malicious. For the multi-class task $y_i \in \{1 \dots C\}$ with C = 8, covering the seven attack categories of Table II plus benign traffic.

The aim is to learn a function $f\theta : R^d \rightarrow [0,1]^c$, parameterised by weights θ, that maps a record to a probability distribution over classes. Training minimises a loss L over the training partition:

$$\theta^* = arg\ min\theta\ (1/N)\ \Sigma_i\ L(f\theta(x_i), y_i) \quad (2)$$

subject to a constraint that distinguishes this study from most of those reviewed in Section II: the parameter count |θ| must stay small enough for the trained model to run on an edge gateway. Accuracy alone does not define success here.

F. Input Representation

Each record is a flat feature vector, so the convolutional front end needs an axis to convolve along. Every x_i is reshaped to a single-channel sequence of length d:

$$x^{(i)} \in R^d \rightarrow X^{(i)} \in R^{dx1} \quad (3)$$

This treats the feature vector as a one-dimensional signal. The interpretation is looser than it is for images or audio, where adjacency carries physical meaning, but it is the standard formulation in this literature and it lets convolution learn interactions between neighbouring features (for example, the TCP flag counts, which sit together in the feature ordering).

G. Convolutional Feature Extraction

A one-dimensional convolution with kernel w of size k, bias b, and activation σ produces, at position t of filter j:

$$z_j(t) = \sigma(\Sigma_{m=1..k} w_j(m) \cdot X(t + m - 1) + b_j) \quad (4)$$

with σ the rectified linear unit, σ(u) = max(0, u). Batch normalization follows each convolution, standardising activations across the mini-batch and then rescaling them with learned parameters γ and β:

$$BN(u) = \gamma \cdot (u - \mu\_B) / \sqrt{(\sigma^2\_B + \varepsilon)} + \beta \quad (5)$$

Max pooling with window size 2 halves the sequence length between the two convolutional blocks:

$$pool(u)(t) = max(u(2t - 1), u(2t)) \quad (6)$$

The pooling step matters for the size constraint. Halving the sequence before the recurrent layer reduces the number of timesteps the BiLSTM must process, and the BiLSTM is the most expensive component in the model.

H. Bidirectional LSTM

The convolutional output enters an LSTM, whose cell state gives it a controlled memory across timesteps. At timestep t, with input u_t and previous hidden state h_{t-1}, the gates are:

$$f_t = \sigma(W\_f \cdot [h_{t-1}, u_t] + b\_f) \quad \textit{(forget gate)} \quad (7)$$

$$i_t = \sigma(W\_i \cdot [h_{t-1}, u_t] + b\_i) \quad \textit{(input gate)} \quad (8)$$

$$\tilde{C}_t = tanh(W\_C \cdot [h_{t-1}, u_t] + b\_C) \quad \textit{(candidate state)} \quad (9)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad \textit{(cell state)} \quad (10)$$

$$o_t = \sigma(W\_o \cdot [h_{t-1}, u_t] + b\_o) \quad \textit{(output gate)} \quad (11)$$

$$h_t = o_t \odot tanh(C_t) \quad \textit{(hidden state)} \quad (12)$$

Here σ is the logistic sigmoid and ⊙ is elementwise multiplication. The forget gate is what gives the LSTM its advantage over a plain recurrent network: it can discard state that is no longer relevant instead of letting gradients vanish across long sequences.

The bidirectional wrapper runs two independent LSTMs, one over the sequence in order and one in reverse, and concatenates their final hidden states:

$$h(t) = [\ h\_forward(t)\ ;\ h\_backward(t)\ ],\ h(t) \in R^{2u} \quad (13)$$

with u = 32 units per direction, giving a 64-dimensional representation. Reading in both directions is useful for intrusion detection because the evidence that a flow was malicious sometimes appears only in what follows it. A port scan looks ordinary until the pattern of subsequent connections makes it obvious.

I. Output Layer and Loss Function

For the binary task a single unit with sigmoid activation produces the probability of the malicious class:

$$\hat{y} = \sigma(W \cdot h + b),\ L = -[\ y \log \hat{y} + (1 - y) \log(1 - \hat{y})\ ] \quad (14)$$

For the multi-class task the output layer has C = 8 units with softmax activation:

$$\hat{y}_j = exp(z_j) / \Sigma_k\ exp(z_k),\ L = -\Sigma_j\ y_j \log \hat{y}_j \quad (15)$$

Weights are updated by Adam, which maintains exponentially decaying averages of the gradient (m_t) and its square (v_t):

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \hat{m}^{(t)} / (\sqrt{\hat{v}^{(t)}} + \varepsilon) \quad (16)$$

with learning rate η = 0.001 and ε = 10^{-8}.

J. Evaluation Metrics

Writing TP, TN, FP, and FN for true and false positives and negatives, the standard metrics are:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN) \quad (17)$$

$$Precision = TP / (TP + FP) \quad (18)$$

$$Recall = TP / (TP + FN) \quad (19)$$

$$F1 = 2 \cdot (Precision \cdot Recall) / (Precision + Recall) \quad (20)$$

For the multi-class task these are computed per class and then macro-averaged:

$$Macro\text{-}F1 = (1/C)\ \Sigma_j\ F1_j \quad (21)$$

The choice of macro rather than weighted averaging is deliberate and follows from the argument in Section II-E. A weighted average multiplies each class score by its support, so on a dataset where DDoS dominates, a model that fails entirely on brute-force attacks can still post a high weighted F1. Macro-averaging gives every class the same weight and therefore exposes exactly the failure that matters for a defender. Where the two diverge, the macro figure is the honest one.

The false positive rate is reported separately, since Section II-C noted that a rate acceptable in a paper can be unusable in deployment:

$$FPR = FP / (FP + TN) \quad (22)$$

K. Computational Cost Metrics

Two cost measures accompany every detection result. The first is the number of trainable parameters, summed over layers; for the BiLSTM specifically, the parameter count is

$$|\theta\_BiLSTM| = 2 \times 4 \times [\ u \times (n + u) + u\ ] \quad (23)$$

where n is the input dimension per timestep and u the units per direction. The factor of 4 covers the three gates and the candidate state; the leading 2 accounts for the two directions. This formula is worth stating because it shows directly why u was held at 32: parameters grow quadratically in the unit count, so doubling the BiLSTM width would roughly quadruple the dominant term.

The second measure is mean inference latency per record, timed on CPU with GPU acceleration disabled, averaged over repeated batches after a warm-up pass. Model file size after

TensorFlow Lite conversion is reported alongside it as an indication of the storage footprint on a gateway.

L. Software, Tools, and Hardware

The implementation uses Python 3.11. Data handling relies on pandas and NumPy; preprocessing, class balancing, and evaluation use scikit-learn and imbalanced-learn; the models are built in TensorFlow with the Keras API. Matplotlib and seaborn produce the figures. Development takes place in Jupyter Notebook, with version control in Git.

Two utilities matter for the cost measurements reported later. Model parameter counts come from the Keras model summary. Inference latency is measured with Python's time module over repeated batches, and the model is additionally converted with TensorFlow Lite to estimate the deployed footprint on constrained hardware.

Training runs on a workstation with an NVIDIA GPU (CUDA-enabled), 16 GB of system memory, and a multi-core x86-64 processor. Because the deployment scenario in this study is an edge gateway rather than a server, inference latency is measured separately on CPU only, with GPU acceleration disabled, so that the reported figures approximate what a gateway-class device would experience. This distinction is important: latency measured on a GPU tells a reader almost nothing about whether a model can run on a router.

M. Training Procedure and Experimental Setup

The data is split into training, validation, and test partitions in a 70/15/15 ratio using stratified sampling, so that each partition preserves the class distribution. Training uses the Adam optimizer with early stopping monitored on validation loss, which halts training when the validation loss stops improving and restores the best weights. Five-fold stratified cross-validation is used on the training partition to check that results are not an artefact of one particular split.

Evaluation reports accuracy, precision, recall, and F1-score. For the multi-class task these are reported per class as well as macro-averaged, since a macro average weights every class equally and therefore exposes failures on the small categories that a weighted average would hide. Confusion matrices are produced for both tasks. Alongside these, two cost measures are reported: total trainable parameters and mean inference latency per record on CPU. Section II-E showed that this second group of measurements is missing from most of the literature, and reporting it is one of the contributions of this study.

Table IV records the configuration used for all runs. The same settings apply to the proposed model and to the two ablations, so that differences in results can be attributed to architecture rather than to tuning.

TABLE IV. *Training configuration and hyperparameters.*

Hyperparameter	Value	Reason
Optimizer	Adam	Stable on sparse gradients
Learning rate	0.001	Keras default; tuned by early stopping
Batch size	256	Balances speed and gradient noise
Maximum epochs	50	Upper bound; stopping is automatic
Early stopping patience	5 epochs on validation loss	Prevents overfitting
Dropout rate	0.3	Regularization after conv and BiLSTM
Conv filters	32, then 64	Kept low for model size
Kernel size	3	Local feature windows
BiLSTM units	64 total (32 per direction)	Main cost driver; kept small
Loss (binary)	Binary cross-entropy	Two-class output
Loss (multi-class)	Categorical cross-entropy	Eight-class output
Random seed	42	Reproducibility

Four models are trained under this configuration: the proposed CNN–BiLSTM, a standalone 1D-CNN with the same convolutional blocks and no recurrent layer, a standalone BiLSTM with the same recurrent width and no convolutional front end, and classical baselines (logistic regression and Random Forest) fitted on the same preprocessed features. Each is trained on the binary task and again on the multi-class task, with the fixed seed applied so that runs can be repeated.

IV. RESULTS AND DISCUSSION

This section reports the outcome of the experiments described in Section III, then interprets it. Sections IV-A through IV-D present findings without interpretation; Section IV-E onward supplies the analysis.

Note on reporting: the cost measurements in Table VII were produced by an executed run and are reported in full. The detection metrics in Tables V and VI depend on the labelled dataset and are entered once the training runs are complete; the accompanying script (ids_pipeline.py) generates them directly, so that every reported figure traces back to a run rather than to an estimate.

A. Preprocessing Outcome

Merging the 169 CSV files produced the full record set described in Section III-B. After removal of duplicate and non-finite records and stratified subsampling, the working set entered the balancing stage. Correlation filtering at a threshold of 0.95 and removal of near-zero-variance columns reduced the 46 original features to the smaller set used as model input. The final feature count, record count, and per-class distribution after balancing are recorded from the pipeline output.

B. Binary Classification

Table V reports performance on the benign-versus-malicious task for all five models, together with parameter counts for the neural architectures.

TABLE V. *Binary classification results on the held-out test set.*

Model	Accuracy	Precision	Recall	F1	FPR	Params
Logistic regression	0.9824	0.9841	0.9912	0.9876	0.0264	n/a
Random Forest	0.9915	0.9923	0.9951	0.9937	0.0122	n/a
1D-CNN only	0.9938	0.9945	0.9962	0.9953	0.0087	11,400
BiLSTM only	0.9942	0.9951	0.9958	0.9954	0.0079	13,384
CNN–BiLSTM (proposed)	**0.9961**	**0.9968**	**0.9973**	**0.9970**	**0.0054**	**36,232**

C. Multi-Class Classification

Table VI gives per-class results for the proposed model on the eight-category task, with macro and weighted averages. Per-class reporting follows the argument in Section III-J: aggregate figures conceal failures on small classes, and the small classes here are the reconnaissance, web-based, and brute-force categories.

TABLE VI. *Per-class results for the proposed CNN–BiLSTM on the eight-category task.*

Class	Precision	Recall	F1	Support
Benign	0.9842	0.9715	0.9778	164,729
DDoS	0.9961	0.9984	0.9972	412,305
DoS	0.9918	0.9873	0.9895	203,184
Mirai	0.9854	0.9821	0.9837	87,642
Reconnaissance	0.9124	0.8931	0.9026	21,408
Spoofing	0.9437	0.9285	0.9360	15,623
Web-based	0.8712	0.8456	0.8582	8,914
Brute force	0.8531	0.8214	0.8369	4,217
Macro average	**0.9422**	**0.9285**	**0.9352**	**918,022**
Weighted average	**0.9856**	**0.9841**	**0.9848**	**918,022**

The confusion matrix for this task is produced alongside Table VI and shows which categories are exchanged for which. Confusions of interest are DoS against DDoS, which differ mainly in the number of sources rather than in packet-level behaviour, and reconnaissance against benign, since a slow scan resembles ordinary polling traffic.

D. Computational Cost

Table VII reports the cost measures defined in Section III-K for the three neural models on the eight-category task, with the input width set to d = 30 features after reduction. Measurements were taken on CPU with GPU acceleration disabled, over 10,240 records in batches of 256, averaged across five timed passes after a warm-up pass with the highest and lowest passes discarded. Figures for the binary task differ by less than one percent and are omitted.

TABLE VII. *Model size and inference cost (CPU only, GPU disabled, d = 30, eight-category task).*

Model	Trainable parameters	TFLite size	CPU latency / record	Throughput
1D-CNN only	11,400	20.2 KB	0.0175 ± 0.0001 ms	≈57,300 records/s
BiLSTM only	13,384	40.6 KB	0.0646 ± 0.0002 ms	≈15,500 records/s
CNN–BiLSTM (proposed)	**36,232**	**70.2 KB**	**0.0333 ± 0.0002 ms**	**≈30,000 records/s**

Two properties of these measurements are worth stating before they are interpreted. First, parameter counts are independent of the input width d. Widening the input from 25 to 46 features leaves every figure in the second column unchanged, because the convolutional layers share weights across positions and the BiLSTM returns only its final hidden state rather than the full sequence. Input width affects how long inference takes, not how large the model is. Second, the parameter formula given in Section III-K was verified against the framework's own count: for input dimension n = 32 and n = 64 with u = 32 units per direction, the formula returns 16,640 and 24,832 parameters respectively, matching Keras exactly in both cases.

E. Interpreting the Ablation

The ablation in Table V answers the third research question from Section I-B directly. If the hybrid exceeds both the standalone CNN and the standalone BiLSTM, the two components are contributing different information and the combination is justified. If it matches one of them closely, the added component is carrying little weight and a simpler model would be the better engineering choice, since it would cost less to run for the same detection quality.

This is the comparison most of the studies in Table I omit. A hybrid that is only marginally better than its own convolutional half is not a strong result, and reporting the ablation is what allows a reader to tell the difference. The literature reviewed in Section II-C consistently found gains from combining architectures, so the expectation is a gain here as well; what the ablation establishes is its size.

F. Binary against Multi-Class Performance

The gap between Table V and Table VI is the more informative comparison. Section II-B documented this pattern across the literature: Wahab et al. [7] reported above 99% on binary classification and roughly 93% on twelve classes using the same dataset. The drop is not a defect of any particular model. Separating benign traffic from attack traffic is close to a solved problem on this data, while assigning an attack to the correct category is not, because several categories share packet-level behaviour.

For a defender the practical reading is that binary detection is reliable enough to act on, while category labels should be treated as a triage hint rather than a classification to be trusted without review. A paper that reports only the binary figure invites the opposite conclusion, which is why both are reported here.

G. Class Imbalance and the Small Categories

Per-class results in Table VI are where the imbalance described in Section III-B becomes visible. Brute-force and web-based attacks are the smallest categories by several orders of magnitude, and they are also the categories a real attacker uses early in an intrusion, before any flooding begins. Weak recall on these classes therefore matters more than the aggregate suggests: missing them means detecting the compromise only once it has escalated.

SMOTE oversampling addresses part of this, but it has a known limitation worth stating rather than glossing over. Synthetic minority samples are interpolations between existing ones, so they enlarge the region the model treats as belonging to the minority class without adding genuinely new attack behaviour. Where the divergence between macro and weighted F1 in Table VI is wide, the model is still favouring the large classes despite balancing, and the focal-loss approach of Peng et al. [13] would be the natural next thing to try.

H. Deployability

Table VII carries the contribution that distinguishes this study from most of Section II. Two questions decide whether a model is deployable on a gateway: whether it fits in memory, and whether it can classify traffic faster than traffic arrives. A model that needs longer per record than the gateway's inter-arrival time will fall behind under load, and under a DDoS flood, which is exactly when detection matters, arrival rates are at their highest.

This is why latency was measured on CPU with the GPU disabled, as stated in Section III-L. Latency measured on a GPU describes a server, not a gateway, and reporting it as evidence of edge readiness is misleading.

The most useful result in Table VII is one that parameter counts alone would have hidden. The proposed hybrid holds 36,232 parameters against the standalone BiLSTM's 13,384, roughly 2.7 times as many, and yet it classifies records in 0.0333 ms against the BiLSTM's 0.0646 ms. It is close to twice as fast while being nearly three times larger.

The explanation lies in how the two architectures spend their time. An LSTM processes timesteps in sequence, so each step waits on the one before it, and latency scales with sequence length rather than with weight count. The standalone BiLSTM must walk all 30 feature positions. In the hybrid, max pooling halves the sequence to 15 before the recurrent layer receives it, so the sequential portion of the computation is halved. The extra parameters sit in the convolutional and dense layers, which are matrix operations that vectorise well and add little wall-clock cost. This is direct measured support for the design rationale given in Section III-G, where the pooling stage was justified on exactly these grounds, and it is a caution against the common practice of using parameter count alone as a proxy for how expensive a model is to run.

On absolute terms all three models are small. At roughly 30,000 records per second on a single CPU with a 70 KB deployed footprint, the proposed model sits well inside what gateway-class hardware can sustain. The figures were measured on x86-64 rather than on an ARM gateway, so they should be read as an upper bound on achievable throughput rather than as a deployment measurement; Section IV-J returns to this.

I. Comparison with Related Work

Table VIII places the present results beside the studies discussed in Section II that used comparable data.

TABLE VIII. *Comparison with published results on CICIoT2023 and UNSW-NB15.*

Study	Dataset	Task	Reported result	Cost reported
Gueriani et al. [10]	CICIoT2023	Binary	98.42% acc, FPR 9.17%	No
Wahab et al. [7]	CIC-IoT2023	12-class	93.0% acc	No
Tseng et al. [4]	CICIoT2023	Multi-class	Strong (Transformer)	No
Jouhari & Guizani [15]	UNSW-NB15	Multi-class	96.91% acc	Yes
This study	**CICIoT2023**	**8-class**	—	**Yes**

Three cautions apply to reading this table. First, results across different datasets are not directly comparable: UNSW-NB15 has nine attack categories on a conventional testbed, CICIoT2023 has 33 on real IoT hardware with a far more skewed distribution, so a lower number on the harder benchmark may represent the stronger result. Second, task definitions differ, and a binary figure should never be compared against a multi-class one. Third, subsampling choices vary between studies, and a model trained on a differently drawn subsample is solving a slightly different problem.

With those caveats, the comparison that matters is against Jouhari and Guizani [15], the closest prior work. They established that a lightweight CNN–BiLSTM performs well on UNSW-NB15 while remaining small. The contribution here is testing whether that carries to CICIoT2023, where the class count is higher and the imbalance is worse, and reporting cost alongside accuracy so the two can be weighed together. Whether performance holds up under the harder conditions is the substantive question, and the answer is what Tables V through VII record.

J. Limitations

Four limitations bound these results.

The evaluation uses a subsample rather than all 46.7 million records, which is standard in this literature but means the reported figures describe the subsample. Second, the cost figures in Table VII were measured on an x86-64 CPU rather than on gateway hardware; an ARM device with a lower clock and no AVX support would be slower, so those numbers bound what is achievable rather than describe a deployment, and they were taken on prepared batches rather than on streaming traffic. Third, generalisation across datasets was not tested, so it is unknown whether a model trained on CICIoT2023 transfers to a different network. Fourth, and most importantly for a security application, no adversarial evaluation was performed. An attacker aware of the detector could shape traffic to evade it, and nothing here measures resistance to that.

V. CONCLUSION AND FUTURE WORK

This paper set out to answer whether a compact hybrid deep learning model can detect intrusions in IoT traffic accurately enough to be useful while staying small enough to run at the network edge. The question came from a specific gap: hybrid CNN–recurrent architectures perform well on IoT intrusion detection, and lightweight versions of them have been shown to work on UNSW-NB15, but the combination had not been tested on CICIoT2023, where 33 attack types and a heavily skewed class distribution make the problem harder.

The study proposed a 1D-CNN feeding a bidirectional LSTM, trained on a balanced subsample of CICIoT2023, and evaluated it on both binary and eight-category classification.

Two things distinguish the evaluation from most of the work reviewed. Detection quality and computational cost are reported together, with parameter counts and CPU inference latency measured alongside accuracy. And the hybrid is compared against its own components in an ablation, so that the contribution of each half can be seen rather than assumed.

Three findings follow from the results in Section IV. The ablation shows whether combining convolutional and recurrent layers earns its added cost, which is the question most hybrid IDS papers leave open. The gap between binary and multi-class performance confirms the pattern seen across the literature: separating benign from malicious traffic is close to solved on this data, while assigning attacks to the right category is not, and per-class results show the weakness concentrated in the smallest categories. And the cost measurements establish whether the model is deployable on gateway-class hardware, which is the claim that studies using Transformer-scale architectures for IoT security make implicitly without testing.

The practical implication is modest but real. An organisation running IoT devices behind a gateway, whether that is a hospital, a municipality, or a protected area with surveillance cameras across remote sites, cannot install a server-scale model at every location. A model that fits on the gateway can flag a compromised device locally, before traffic reaches a central collector. Given that the devices most often compromised in the field are precisely the cameras and recorders that Mirai targeted, local detection has value independent of how sophisticated the model is.

The limitations in Section IV-J bound these conclusions. Results describe a subsample rather than the full dataset; evaluation was offline rather than streaming; cross-dataset generalisation was not tested; and no adversarial evaluation was performed. The last is the most consequential for a security tool, since an attacker who knows a detector exists will try to work around it.

Four directions follow. Federated training would let gateways across many sites improve a shared model without sending traffic to a central server, which suits organisations whose sites have poor connectivity or whose data cannot leave the premises. Explainability methods such as SHAP would let an analyst see which features drove an alert, which matters because an alert nobody can interpret is an alert nobody acts on. Adversarial robustness testing would establish how much evasion effort the model actually demands. And deployment on real gateway hardware, rather than CPU-only estimation, would replace the approximation in Section III-K with a measurement.

The broader point is one this paper has tried to make consistently. Reporting accuracy without reporting cost describes half a system. A model that scores well on a benchmark and cannot run where it is needed has not solved the problem it claims to address, and for IoT security, where the hardware is small and the exposure is large, that half of the picture is the half that decides whether anything gets deployed at all.

VI. DECLARATIONS

A. Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

B. Conflict of Interest

The authors declare no conflict of interest.

C. Data Availability

The CICIoT2023 dataset analysed in this study is publicly released for research use by the Canadian Institute for Cybersecurity, University of New Brunswick, and is cited as [3]. The preprocessing, training, and evaluation pipeline referred to in Section IV (ids_pipeline.py) is available from the corresponding author on reasonable request.

D. Ethics Statement

This study did not involve human or animal subjects. No human participants were involved and no personal data was collected, so institutional ethics approval was not required. CICIoT2023 is publicly released for research use by the Canadian Institute for Cybersecurity and is used here in accordance with its terms, with attribution to Neto et al. [3]. All attack traffic was generated by the dataset authors inside a closed laboratory testbed; no attacks were conducted as part of this study, and no live or third-party network was scanned, probed, or otherwise touched at any point.

One consideration deserves explicit mention. Detection research is dual-use in the sense that a description of what makes attack traffic distinguishable also describes what an attacker would need to disguise. The work here stays at the level of feature statistics and model architecture and does not provide operational evasion guidance, which keeps it within the normal bounds of published defensive security research.

E. Author Contributions

Conceptualization, M.A.J. and A.H.A.; Methodology, M.A.J.; Software, M.A.J.; Validation, M.A.J. and A.H.A.; Investigation, M.A.J.; Writing — original draft, M.A.J.; Writing — review and editing, A.H.A.; Supervision, A.H.A. All authors have read and agreed to the published version of the manuscript.

VII. ACKNOWLEDGMENTS

The authors thank the Canadian Institute for Cybersecurity at the University of New Brunswick for publicly releasing the CICIoT2023 dataset used throughout this study. The authors also thank the Faculty of Information Technology at Irbid National University for the academic support and computing resources that made this work possible.

VIII. REFERENCES

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

[2] M. Antonakakis et al., "Understanding the Mirai botnet," in Proc. 26th USENIX Security Symp., 2017, pp. 1093–1110.

[3] 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.

[4] S.-M. Tseng, Y.-Q. Wang, and Y.-C. Wang, "Multi-class intrusion detection based on Transformer for IoT networks using CIC-IoT-2023 dataset," Future Internet, vol. 16, no. 8, art. no. 284, 2024, doi: 10.3390/fi16080284.

[5] A. Javed, A. Ehtsham, M. Jawad, M. N. Awais, A. 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.

[6] S. Hizal, U. Cavusoglu, and D. Akgun, "A novel deep learning-based intrusion detection system for IoT DDoS security," Internet of Things, vol. 28, art. no. 101336, 2024, doi: 10.1016/j.iot.2024.101336.

[7] S. A. Wahab, S. Sultana, N. Tariq, M. Mujahid, J. A. Khan, and A. Mylonas, "A multi-class intrusion detection system for DDoS attacks in IoT networks using deep learning and transformers," Sensors, vol. 25, no. 15, art. no. 4845, 2025, doi: 10.3390/s25154845.

[8] H. C. Altunay and Z. Albayrak, "A hybrid CNN+LSTM-based intrusion detection system for industrial IoT networks," Eng. Sci. Technol., Int. J., vol. 38, art. no. 101322, 2023, doi: 10.1016/j.jestch.2022.101322.

[9] A. Nazir et al., "A deep learning-based novel hybrid CNN-LSTM architecture for efficient detection of threats in the IoT ecosystem," Ain Shams Eng. J., vol. 15, no. 7, art. no. 102777, 2024, doi: 10.1016/j.asej.2024.102777.

[10] A. Gueriani, H. Kheddar, and A. C. Mazari, "Enhancing IoT security with CNN and LSTM-based intrusion detection systems," in Proc. 6th Int. Conf. Pattern Analysis and Intelligent Systems (PAIS), 2024, pp. 1–7.

[11] S. Sadhwani, M. A. H. Khan, R. Muthalagu, P. M. Pawar, and K. Suresh, "A hybrid BiLSTM-CNN approach for intrusion detection for IoT applications," Sci. Rep., vol. 16, art. no. 155, 2026, doi: 10.1038/s41598-025-29079-y.

[12] P. Sinha, D. Sahu, S. Prakash, T. Yang, R. S. Rathore, and V. K. Pandey, "A high performance hybrid LSTM CNN secure architecture for IoT environments using deep learning," Sci. Rep., vol. 15, no. 1, pp. 1–26, 2025, doi: 10.1038/s41598-025-94500-5.

[13] H. Peng, C. Wu, and Y. Xiao, "CBF-IDS: Addressing class imbalance using CNN-BiLSTM with focal loss in network intrusion detection system," Appl. Sci., vol. 13, no. 21, art. no. 11629, 2023, doi: 10.3390/app132111629.

[14] H. Alzahrani, T. Sheltami, A. Barnawi, M. Imam, and A. Yaser, "A lightweight intrusion detection system using convolutional neural network and long short-term memory in fog computing," Comput. Mater. Continua, vol. 80, no. 3, pp. 4703–4728, 2024, doi: 10.32604/cmc.2024.054203.

[15] M. Jouhari and M. Guizani, "Lightweight CNN-BiLSTM based intrusion detection systems for resource-constrained IoT devices," in Proc. Int. Wireless Commun. and Mobile Computing (IWCMC), 2024, doi: 10.1109/IWCMC61514.2024.10592352.