IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 59, No. 10, October, 2026, pp. 2903-2916
Journal homepage: http://iieta.org/journals/jesa

Attention-Enhanced Lightweight Convolutional Network with Pruning and Quantization for Efficient Multi-Vector Internet of Things Malware Detection

Andi Novianto[1,2*], Fatchul Arifin[1,3], Herman Dwi Surjono[1,3], Mochamad Subchan Mauludin[4], Singgih Dwi Prasetyo[5]

[1] The Doctoral Program on Engineering Science, Department of Electrical Engineering, Faculty of Engineering, Universitas Negeri Yogyakarta, Yogyakarta 55281, Indonesia
[2] Faculty of Computer Science, Universitas Duta Bangsa Surakarta, Surakarta 57154, Indonesia
[3] Department of Engineering Science, Universitas Negeri Yogyakarta, Yogyakarta 55281, Indonesia
[4] Department of Informatics Engineering, Universitas Wahid Hasyim, Semarang 50236, Indonesia
[5] Power Plant Engineering Technology, State University of Malang, Malang 65145, Indonesia

Corresponding Author Email: andi_novianto@udb.ac.id

Copyright: ©2026 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

https://doi.org/10.18280/jesa.591012

Received: 27 August 2026
Revised: 23 September 2026
Accepted: 3 October 2026
Available online: 31 October 2026

Keywords:
attention mechanism, edge artificial intelligence, Internet of Things security, malware detection, model compression, quantization

ABSTRACT

Resource-constrained Internet of Things (IoT) devices require malware detectors that maintain strong predictive performance while limiting computation and memory demand. Hence, this study treats the trade-off between detection quality and measured computational cost, rather than physical edge deployment, as its primary objective. An attention-enhanced lightweight convolutional network was evaluated on four reported traffic categories (normal, distributed denial-of-service (DDoS), remote code execution (RCE), and application programming interface (API) exploitation or data exfiltration), using heterogeneous public and controlled records that were harmonized into 128-token attribute-value sequences, embedded, processed by a one-dimensional convolutional extractor, aggregated with attention, and compressed by pruning and 8-bit quantization. The reported test accuracy was 98.45% with a weighted precision of 0.98, a recall of 0.99, and an F1-score of 0.985; removing attention lowered the F1-score to 0.975, removing pruning or quantization increased processor and memory demand with only marginal predictive change, and the study-specific efficiency score of 1.49 exceeded those of ClassicCNN and long short-term memory (LSTM) baselines. A recomputation under a common normalization ranked a TinyML-oriented baseline higher, weighted recall was not fully consistent with accuracy, and hardware, repeated-run uncertainty, compression schedule, per-class results, and class accounting were not documented, so the findings are resource-oriented and require independent measurement of latency, power, and throughput on an edge device before real-time readiness is claimed.

1. INTRODUCTION

The Internet of Things (IoT) now supports consumer, industrial, healthcare, mobility, and infrastructure services, but its heterogeneous hardware, long device lifecycles, and extensive network exposure enlarge the attack surface [1, 2]. Weak authentication, insecure messaging, vulnerable application programming interfaces (APIs), and exposed management services can facilitate distributed denial-of-service (DDoS), remote code execution (RCE), credential abuse, and data exfiltration [3-6]. Signature-based network intrusion detection remains valuable for known patterns, yet it is brittle when malicious code is polymorphic, previously unseen, or expressed through variable traffic sequences [7-10].

Machine-learning and deep-learning detectors can learn discriminative structure directly from network flows or behavioral sequences. Traffic classifiers and convolutional, recurrent, and transformer-based intrusion detectors have achieved strong results on established IoT datasets [11-14]. Nevertheless, reported accuracy alone is insufficient for deployment on gateways, microcontrollers, or single-board computers. A usable detector must control latency, processor load, memory footprint, and model size; preserve sensitivity to rare or high-impact attacks; and provide enough interpretability for security analysts to inspect decisions.

Attention mechanisms improve feature selection by assigning greater weight to informative spatial, channel, or temporal representations [15-18]. However, their additional parameters can erode the efficiency of lightweight architectures. Model compression through pruning and low-precision quantization offers a complementary strategy: redundant connections are removed, and compact integer representations replace high-precision weights [19, 20]. The central design problem is therefore not simply to maximize accuracy, but to optimize a measurable trade-off among detection quality, computational demand, and deployment constraints.

The primary objective of this study is to quantify the trade-

off between detection quality and computational cost for an attention-enhanced Lightweight Convolutional Neural Network (LightCNN) trained on a reported multi-source IoT corpus, with detection accuracy treated as a quality constraint, computational cost as the quantity to be reduced, and edge deployment as the motivating context rather than a validated outcome, as outlined in the workflow of Figure 1. The experimental record includes ablations of attention, pruning, and quantization; comparisons with ClassicCNN and long short-term memory (LSTM) baselines; and additional comparisons with MobileNetV2 and a TinyML-oriented convolutional baseline. The original numerical results are retained when traceable, but ambiguous claims are corrected: 98.45% is used consistently for the reported LightCNN accuracy; elapsed experiment time is not mislabeled as per-sample latency; and the performance component of the efficiency score is documented as the arithmetic mean of accuracy, precision, recall, and F1-score. Accordingly, the contributions listed below concern the detector design, the attribution of accuracy and cost to individual components, and a transparent cost index. In contrast, measured latency, power, and throughput on edge hardware were not obtained and are not claimed.

The contributions are fourfold:

•A reproducible conceptual pipeline for harmonizing multi-source IoT traffic into fixed-length token sequences and classifying them with an attention-enhanced one-dimensional LightCNN.

•A component-level ablation analysis that separates the predictive contribution of attention from the computational contributions of pruning and quantization.

•A transparent performance-to-cost formulation that combines predictive quality with normalized elapsed time, CPU use, and memory use.

•An evidence reconciliation and validity assessment that identifies missing metadata, class-accounting inconsistency, and the edge-device measurements required before operational deployment claims are made.

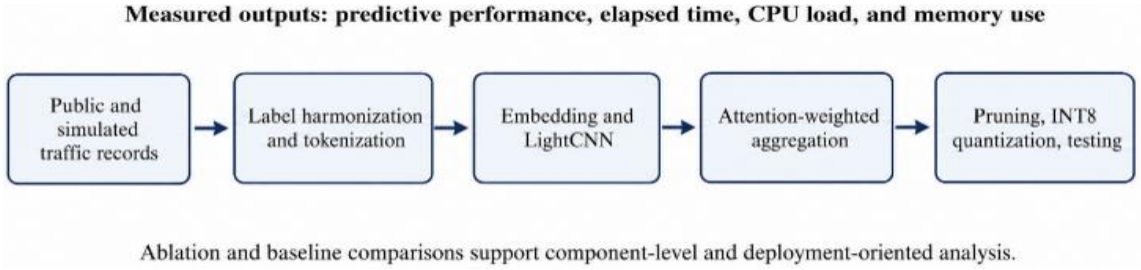


Figure 1. End-to-end experimental workflow (author-prepared schematic)

2. RELATED WORK AND COMPARATIVE POSITIONING

2.1 Internet of Things intrusion and malware detection

IoT security research has progressed from manually curated signatures toward statistical and representation-learning approaches. Machine-learning classifiers remain attractive because of their moderate computational demand, but their effectiveness depends on feature engineering and may deteriorate when sequential relationships or previously unseen behaviors dominate [21-24]. Deep models learn local, temporal, and global dependencies more directly, as illustrated by transformer-based detection on CICIoT2023, generative modeling on IoT-23, and a convolutional-recurrent ensemble for IoT malware detection [25-28]. In contrast, API-sequence modeling shows that behavioral traces can carry discriminative information not visible in aggregate flow statistics, whereas recent journal studies on lightweight IoT intrusion detection increasingly report compact convolutional or temporal designs and quantization [29, 30].

2.2 Attention, explainability, and compression

Attention-based hybrid detectors combine convolutional extraction with recurrent or transformer context modeling. Recent studies report gains from bidirectional recurrent attention, CNN-LSTM integration, hybrid transformer-CNN structures, two-stage channel-spatial attention, and explainable attention fusion [23, 31]. These gains are not cost-free: Squeeze-and-Excitation and Convolutional Block Attention Module designs demonstrate that compact channel and spatial attention can be effective, whereas MobileNetV2, microcontroller-oriented neural architecture research, and TinyML-oriented studies provide established efficiency baselines, and a scoping review of lightweight intrusion detectors reports that target-device latency, runtime memory, and energy are reported inconsistently [32-35]. Pruning and integer quantization reduce model complexity and memory bandwidth. Still, their deployment value depends on hardware-compatible sparsity and calibrated or quantization-aware conversion. Because recent lightweight detectors differ in datasets, splits, and hardware, Table 1 lists the data sources and their evidence status, and the comparison is kept qualitative.

2.3 Research gap

The evidence synthesis supplied with the source material identifies four recurrent gaps: limited evaluation across heterogeneous datasets, incomplete analysis of compression-induced accuracy loss, insufficient interpretability, and weak real-device validation. The present work is positioned as a resource-oriented detection study that addresses the first two gaps through multi-source harmonization and component ablation, and partially addresses interpretability through attention-based aggregation. It does not resolve weak real-device validation, adversarial robustness, or repeated-run uncertainty, so detection accuracy, computational cost, and edge deployment are related here as a quality constraint, the optimized quantity, and a future validation target, respectively. These remaining gaps are incorporated into the discussion rather than obscured by unsupported claims.

Table 1. Comparative positioning of representative attention-based Internet of Things (IoT) security studies

Source	Type	Reported Role	Evidence Status	Required Reproducibility Metadata
CICIoT2023/CIC-IoT	Public IoT attack traffic	Normal and DDoS-related flows	Verified public dataset and paper [9, 25]	Confirm exact version and selected attack labels
IoT-23/CTU captures	Public benign and malware traffic	DDoS- and RCE-related behavior	Verified dataset release [10, 26]	Provide capture-to-class mapping script
Infiltration.csv	External data file reported in the source	Data-exfiltration-related records	Persistent identifier not supplied	Add repository URL, DOI, license, and checksum
Controlled API simulation	Locally generated records	API exploitation and/or exfiltration	Experiment-specific	Provide scenario scripts, endpoints, seed, and ethical controls
gabungan.csv	Merged working corpus	All reported classes	Experiment-specific derived file	Provide checksum and schema dictionary

Note: The comparison is qualitative because the studies use different datasets, splits, hardware, and class definitions; numerical scores are therefore not treated as directly interchangeable. Application programming interfaces (APIs), distributed denial-of-service (DDoS), remote code execution (RCE).

3. METHODOLOGY

3.1 Study design and data provenance

The reported corpus combines public IoT traffic sources with controlled API-exploitation simulations. CICIoT2023 provides large-scale labeled IoT attack traffic, whereas IoT-23 contains benign and malware capture scenarios with Zeek-derived connection records [36, 37]. The source experiment additionally references selected CTU-IoT malware captures, an infiltration-oriented data file, and simulated API-level attacks. Because the submitted records do not provide persistent identifiers, per-source sample counts, or post-deduplication counts for every transformed file, Table 2 separates verified public provenance from experiment-specific artifacts and maps each source to the class it supplies (Figure 2). In contrast, the per-source class counts and the final split for each source remain to be confirmed by the authors.

Table 2. Dataset provenance and evidence status

Source	Type	Reported Role	Evidence Status	Required Reproducibility Metadata
CICIoT2023/CIC-IoT	Public IoT attack traffic	Normal and DDoS-related flows	Verified public dataset and paper [9, 25]	Confirm exact version and selected attack labels
IoT-23/CTU captures	Public benign and malware traffic	DDoS- and RCE-related behavior	Verified dataset release [10, 26]	Provide capture-to-class mapping script
Infiltration.csv	External data file reported in the source	Data-exfiltration-related records	Persistent identifier not supplied	Add repository URL, DOI, license, and checksum
Controlled API simulation	Locally generated records	API exploitation and/or exfiltration	Experiment-specific	Provide scenario scripts, endpoints, seed, and ethical controls
gabungan.csv	Merged working corpus	All reported classes	Experiment-specific derived file	Provide checksum and schema dictionary

Note: Internet of Things (IoT), application programming interface (API), distributed denial-of-service (DDoS), remote code execution (RCE).

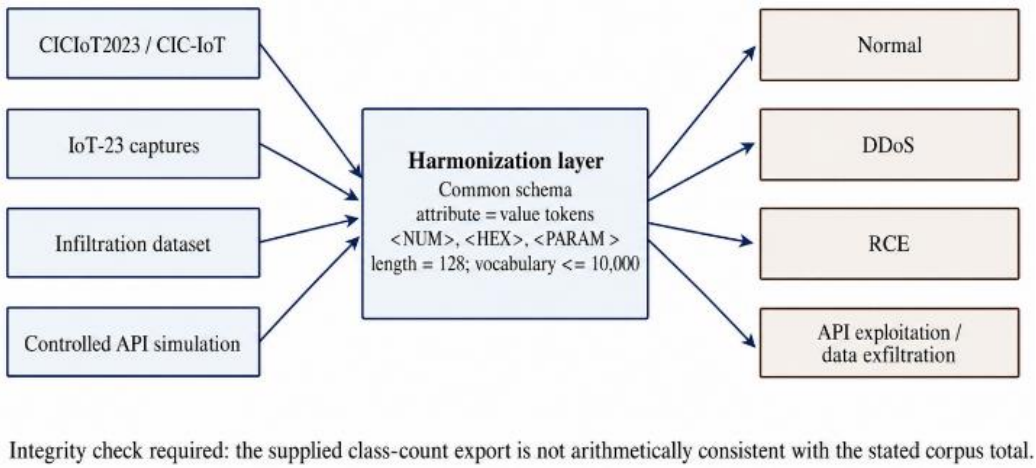


Figure 2. Reported dataset provenance and label-harmonization process (illustrative author-prepared schematic)

3.2 Preprocessing and integrity checks

Categorical and sequential attributes were converted into attribute-value tokens, and numerical, hexadecimal, and variable parameters were normalized as <NUM>, <HEX>, and <PARAM> to reduce sparsity. The vocabulary was capped at 10,000 entries, with padding and unknown-token markers, and each token sequence was padded or truncated to 128 tokens. A trainable 128-dimensional embedding mapped each token identifier to a vector, yielding a 128 × 128 input

matrix for the convolutional layer. The field order used to form a sequence and whether a sequence represents a packet, a flow, or a session were not specified in the source record and are listed as open items in the model specification record. The records were divided by stratified sampling with random seed 42 into 70% training, 15% validation, and 15% testing partitions, and Tables 3 and 4 report the resulting sample accounting and the class prevalence derived from the reported counts.

The split counts are internally consistent, but the class-count export is not. As summarized in Table 4, the four counts indicate that RCE accounts for approximately 97.9% of the records. In contrast, normal, DDoS, and API-exploitation records account for approximately 0.84%, 0.47%, and 0.80%, an imbalance of about 210:1 between the largest and smallest classes. Consequently, class prevalence, weighting strategy, and leakage controls cannot be independently reconstructed from the manuscript alone, and the shares in Table 4 are approximate because the listed counts exceed the stated corpus total by 92,576 records. Before submission, the preprocessing script should generate a single machine-readable audit containing mutually exclusive label counts before and after deduplication, group-aware split identifiers, source-file checksums, and the exact treatment of overlapping scenarios.

Table 3. Reported sample accounting and integrity reconciliation

Item	Reported Value	Interpretation
Stated corpus total	11,484,354	Primary total reported in methodology
RCE count	11,333,427	Reported class count
Normal count	96,949	Reported class count
DDoS count	53,978	Reported class count
API exploitation count	92,576	Reported separately, the arithmetic relationship to the total is unresolved
Sum of all four listed counts	11,576,930	Exceeds stated total by 92,576
Train/validation/test	8,039,047/1,722,653/1,722,654	Sums exactly to 11,484,354

Note: The four listed class counts are not mutually consistent with the stated total. The revision does not alter the numerical record; the authors must verify whether the 92,576 API-exploitation records overlap another class, were excluded from the split, or reflect a labeling/export error. Application programming interfaces (APIs), distributed denial-of-service (DDoS), remote code execution (RCE).

Table 4. Class prevalence and approximate test-set support derived from the reported counts

Class	Source(s) with a Reported Role in Table 2	Reported Count	Share of Listed Total (%)	Approx. Test Support
Remote code execution	IoT-23/CTU captures	11,333,427	97.90	≈1,686,421
Normal	CICIoT2023/CIC-IoT	96,949	0.84	≈14,426
DDoS	CICIoT2023/CIC-IoT; IoT-23/CTU captures	53,978	0.47	≈8,032
API exploitation or data exfiltration	Infiltration.csv; controlled API simulation	92,576	0.80	≈13,775
Sum of the four listed counts	Not applicable	11,576,930	100.00	≈1,722,654

Note: Application programming interfaces (APIs), distributed denial-of-service (DDoS).

3.3 Attention-enhanced lightweight convolutional network

For a token sequence $x = (x_1, ..., x_T)$, the embedding layer maps each token to a vector e_t. A one-dimensional convolution with kernel width k produces local feature vectors h_t. The source record does not specify the number of filters F, the kernel width k, the stride, the padding, pooling, or dropout, and these quantities are therefore kept as symbols in Eq. (1) and in the layer-wise accounting below rather than replaced by invented values. With T = 128 tokens and 128-dimensional embeddings, the embedding matrix contains at most 10,000 × 128 = 1,280,000 trainable parameters, which is the only parameter count that can be derived from the reported specification, and the convolution, attention, and classifier terms are expressed as functions of F, k, and the attention dimension in the layer-wise accounting below.

$$h_t = \mathrm{ReLU}(W_c \cdot e_{t:t+k-1} + b_c) \tag{1}$$

An additive attention layer scores each local feature, normalizes the scores across the sequence, and forms a context vector. Concretely, the attention layer receives the sequence of Conv1D outputs ht, computes a score for every position t through Eq. (2), normalizes the scores across positions with a softmax, and returns a single context vector c as the weighted sum of h_t in Eq. (3). Because the context vector c has the same dimension as one convolutional feature vector, the attention stage adds only the parameters W_a, b_a, and v, and the classifier of Eq. (3) maps c to the four class probabilities. This mechanism allows the classifier to emphasize locally extracted API or traffic patterns that contribute most strongly to the predicted class.

$$u_t = tanh(W_a h_t + b_a), \alpha_t = \frac{exp(v^\top u_t)}{\sum_j exp(v^\top u_j)} \tag{2}$$

$$c = \sum_t \alpha_t h_t, p(y \mid x) = softmax(W_o c + b_o) \tag{3}$$

The model is trained by minimizing categorical cross-entropy. For C classes and N observations, the objective is:

$$L = -\frac{1}{N}\sum_i \sum_c y_{i,c} \log(p_{i,c}) \tag{4}$$

Table 5 records which specification items remain unreported, and, as summarized in Table 6, the Conv1D layer contributes 128kF + F parameters and $T' \cdot k \cdot F \cdot 128$ multiply-accumulate operations, where T' is the output length that depends on the unreported padding and stride settings, and the additive attention layer adds $F \cdot d_a + 2d_a$ parameters for an

unreported attention dimension d_a. The classifier adds 4F + 4 parameters, and the embedding layer adds no multiply-accumulate operations because it is a table lookup that contributes at most 1,280,000 parameters. Table A2 in the Appendix lists the processing steps from tokenization to INT8 conversion, with F, k, and d_a as configuration symbols that must be replaced with the values recorded in the trained model summary before parameters, MACs, and FLOPs can be reported numerically. Figure 3 shows the corresponding functional architecture.

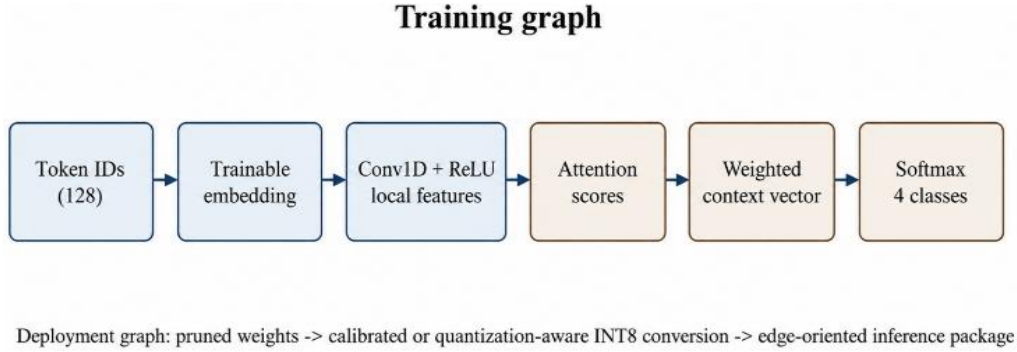


Figure 3. Functional architecture of the attention-enhanced Lightweight Convolutional Neural Network (LightCNN) and deployment compression path (illustrative author-prepared schematic)

Table 5. Model specification and reproducibility status

Component	Current Specification	Status	Required Author Verification
Input sequence	Length 128	Reported	Confirm whether packets, flows, or sessions define one sequence
Vocabulary	Maximum 10,000 tokens	Reported	Provide frequency threshold and out-of-vocabulary rate
Embedding	128 dimensions, trained from scratch	Reported	Provide initializer and regularization
Conv1D block	Convolution + ReLU	Partially reported	Provide filters, kernels, stride, padding, pooling, and dropout
Attention	Additive weighted aggregation	Functional form reported	Provide attention dimension and initialization
Classifier	Softmax, four reported categories	Reported conceptually	Resolve class-accounting discrepancy
Training	Categorical cross-entropy; Adam suggested in the source pseudocode	Incomplete	Provide optimizer, learning rate, batch size, epochs, and early stopping
Pruning	Weight pruning	Incomplete	Provide a structured/unstructured method, schedule, and target sparsity
Quantization	8-bit integer deployment target	Incomplete	State QAT or PTQ, calibration set, operator coverage
Complexity	Parameters, MACs/FLOPs, file size	Not reported	Measure pre- and post-compression values

Table 6. Layer-wise parameter and multiply-accumulate (MAC) expressions of the attention-enhanced Lightweight Convolutional Neural Network (LightCNN)

Layer (Reported Setting)	Output Shape	Parameters	MACs per Sequence
Input token identifiers (T = 128)	128	0	0
Embedding (V ≤ 10,000, d = 128)	128 × 128	≤1,280,000	0 (table lookup)
Conv1D + ReLU (F, k, stride, padding unreported)	$T' \times F$	128kF + F	$T' \cdot k \cdot F \cdot 128$
Additive attention (d_a unreported)	F (context vector)	$F \cdot d_a + 2d_a$	$T'(F \cdot d_a + d_a + F)$
Softmax classifier (4 classes)	4	4F + 4	4F
Total	4	≤1,280,000 + 128kF + $F \cdot d_a + 2d_a$ + 5F + 4	Sum of the MAC terms above

3.4 Pruning and quantization

Pruning removes weights or structures that contribute little to the learned function. Unstructured pruning can achieve high sparsity but may not accelerate inference unless the target runtime supports sparse kernels; structured channel or filter pruning is generally more hardware compatible [17, 18, 21]. Quantization converts weights and, where supported, activations to low-precision representations, and 8-bit integer storage reduces each quantized tensor to one quarter of its 32-bit floating-point size before any pruning is counted. Integer-only inference can reduce memory bandwidth and improve throughput, while quantization-aware training usually preserves accuracy better than naive post-training conversion when activation ranges are difficult to calibrate [19-21].

The source record identifies pruning and INT8 quantization but does not specify whether pruning was structured, the target sparsity, or whether quantization used calibration or quantization-aware training. The procedure that can be stated with confidence is sequential: weight pruning followed by

INT8 conversion, as in the deployment path of Figure 3, whereas the pruning ratio, pruning schedule, post-pruning fine-tuning, calibration data, and operator coverage are not recorded. Model size and parameter count before and after compression cannot be compared numerically because neither was reported; therefore, the component ablation in Section 4.1 compares accuracy, elapsed time, CPU usage, and memory usage between the full model and the variants without pruning or quantization. The revised analysis, therefore, interprets the reported CPU and memory differences empirically, without attributing them to a particular compression algorithm beyond the information supplied.

3.5 Baselines and experimental protocol

All results are reported from the source experiment. The hardware platform, framework versions, batch sizes used for resource measurement, warm-up procedure, number of repeated runs, optimizer settings, learning rate, batch size, number of epochs, learning-rate schedule, and early-stopping rule are not specified. The source record does not confirm that ClassicCNN, LSTM, MobileNetV2, and TinyML-CNN used the same data partition (seed 42), training strategy, and hardware as LightCNN. Accordingly, Table 7 states which conditions are documented, the elapsed-time values are treated as experiment-level measurements rather than per-sample inference latency, and the comparisons in the Results section are interpreted as comparisons between architectural families under partly undocumented conditions. A submission-ready replication should report processor model, memory, operating system, accelerator, software stack, power mode, thread count, batch size, warm-up iterations, repeated runs, mean, standard deviation, and confidence intervals.

As summarized in Table 8, ClassicCNN and LSTM serve as local-pattern and temporal-sequence baselines, whereas MobileNetV2 and TinyML-CNN serve as established efficiency references [38-41]. MobileNetV2 was originally designed for two-dimensional image inputs, so its use on embedded token sequences requires a one-dimensional adaptation of the inverted-residual and depthwise-separable blocks. The source record does not state how this adaptation, the width multiplier, or the initialization were configured. The structures of ClassicCNN (two Conv1D stages) and LSTM (64 recurrent units) are reported only conceptually, and the architecture and runtime of TinyML-CNN are not specified. The comparisons reported in the Results section are therefore interpreted as comparisons between model families evaluated on the same reported metrics rather than as controlled benchmarks of identical training pipelines.

Table 7. Experimental and training conditions and their documentation status

Condition	LightCNN (Proposed)	Baselines (ClassicCNN, LSTM, MobileNetV2, TinyML-CNN)
Data partition	Stratified sampling, seed 42, 70/15/15 (reported)	The identity of the partition is not confirmed
Input representation	128 tokens, 128-dimensional embedding (reported)	Not documented; MobileNetV2 adaptation to token sequences unspecified
Loss function	Categorical cross-entropy (reported)	Not documented
Optimizer	Adam suggested in the source pseudocode	Not documented
Learning rate, batch size, epochs	Not reported	Not reported
Learning-rate schedule, early stopping	Not reported	Not reported
Pruning and quantization	Pruning type, ratio, and schedule; PTQ or QAT not reported	Not documented
Hardware and software	Not specified	Not specified
Repeated runs and variability	Single run; no standard deviation or confidence interval	Single run; no standard deviation or confidence interval
Reported metrics	Accuracy, weighted precision, recall, F1-score, elapsed time, CPU, memory	The same metrics are reported

Note: Lightweight Convolutional Neural Network (LightCNN), long short-term memory (LSTM).

Table 8. Baseline roles and documentation requirements

Model	Reported Structure	Purpose	Reproducibility Note
ClassicCNN	Two Conv1D stages reported conceptually	Local pattern baseline	Higher capacity; no attention
LSTM	64-unit recurrent layer reported in the source table	Temporal-sequence baseline	Higher sequential computation
MobileNetV2	Inverted residual/depthwise-separable architecture [23]	Established mobile baseline	Adaptation to one-dimensional tokens must be documented
TinyML-CNN	Compact convolutional baseline	Microcontroller-oriented efficiency comparison [24]	Exact architecture and runtime required
LightCNN full	Conv1D + attention + pruning + INT8 quantization	Proposed model	Primary subject of ablation

Note: Lightweight Convolutional Neural Network (LightCNN), long short-term memory (LSTM).

3.6 Evaluation metrics and efficiency score

Accuracy, weighted precision, weighted recall, and weighted F1-score summarize predictive performance. Because the reported corpus is highly imbalanced, weighted averages are dominated by the majority class. They should be accompanied by per-class precision, recall, and F1-score, confusion matrices, and macro averages, none of which were

recorded in the source experiment. For a single-label multiclass classifier, support-weighted recall is mathematically identical to accuracy. Hence, any difference between the two beyond rounding indicates a computation or reporting inconsistency that must be resolved before the recall values are interpreted. Precision-recall analysis is generally more informative than receiver operating characteristic analysis in the presence of strong imbalance [27, 28].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP}, Recall = \frac{TP}{TP + FN} \tag{6}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{7}$$

The source table can be reproduced only when the predictive component is defined as the arithmetic mean, not the product, of the four reported metrics:

$$Perf = \frac{Accuracy + Precision_w + Recall_w + F1_w}{4} \tag{8}$$

For model m, elapsed time, CPU usage, and memory usage are normalized to the maximum observed across the compared models. This normalization makes each cost term dimensionless and bounded by 1. Still, it also means that a model's score depends on which models are included in the comparison set, as the recomputation in the results section shows. The weights of 0.60, 0.25, and 0.15 are author-defined priorities that place the largest emphasis on elapsed time, and the source record does not document an empirical or operational derivation of these values. The cost and efficiency scores are therefore defined as:

$$Cost_m = 0.60\left(\frac{Time_m}{Time_{max}}\right) + 0.25\left(\frac{CPU_m}{CPU_{max}}\right) + 0.15\left(\frac{Memory_m}{Memory_{max}}\right) \tag{9}$$

$$Efficiency_m = \frac{Perf_m}{Cost_m} \tag{10}$$

This score is a study-specific index rather than a standardized deployment metric. It is useful for within-experiment ranking but should not be compared across studies unless the same hardware, measurement protocol, normalization set, and weights are used. Its performance term is an arithmetic mean of accuracy, weighted precision, weighted recall, and weighted F1-score, which are strongly correlated and, for single-label classification, include two quantities (accuracy and weighted recall) that are mathematically equal, so predictive quality is partly counted twice. The index is therefore reported alongside a weight-sensitivity analysis and should not be treated as a universal standard for deployment evaluation.

4. RESULT

4.1 Component ablation

Removing attention caused the largest predictive degradation: accuracy fell by 0.87 percentage points, and weighted F1-score fell from 0.985 to 0.975 (Figure 4). This result supports attention as the principal predictive component, at a recorded cost of only 0.5% additional elapsed time, 0.05 percentage points of CPU use, and 3.33 MB of memory. In contrast, removing pruning or quantization changed accuracy by less than 0.2 percentage points but increased resource use, as recorded in Table 9: without pruning, CPU use increased from 9.90% to 14.72% and memory use from 1090.64 MB to 1328.90 MB, and without quantization, CPU use increased to 12.83% and memory use to 1295.22 MB. Relative to these variants, the full model reduced CPU use by 32.7% and 22.8% and memory use by 17.9% and 15.8%, respectively, as derived in Table 10, while its accuracy was 0.13 and 0.06 percentage points higher, indicating that pruning and quantization primarily improve computational efficiency. However, a single run cannot separate such small differences in accuracy from run-to-run variation.

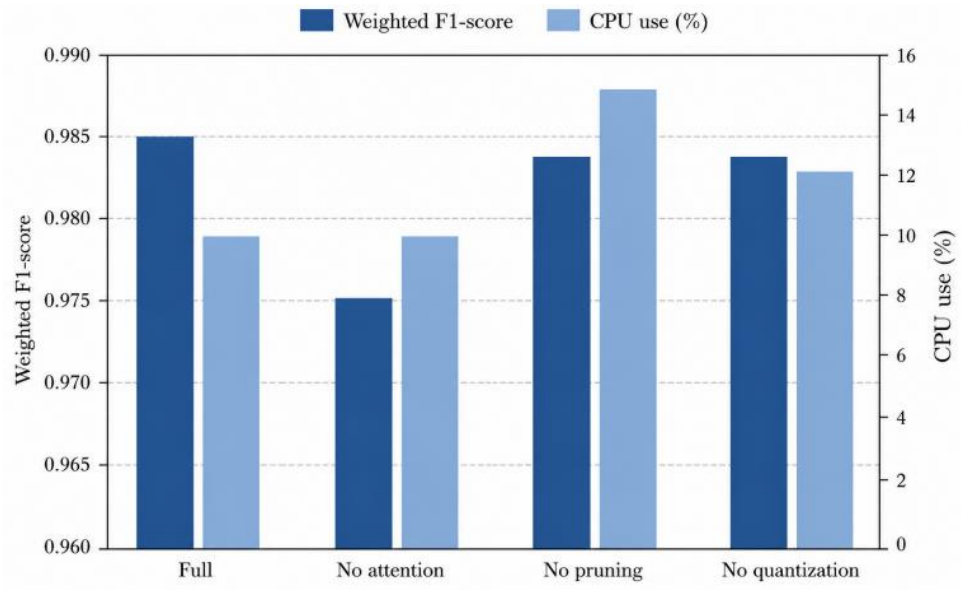


Figure 4. Ablation effects on weighted F1-score and reported CPU use (author-generated visualization from the supplied results)

Table 9. Ablation results for attention, pruning, and quantization

Variant	Accuracy	Precision_w	Recall_w	F1_w	Elapsed Time (s)	CPU (%)	Memory (MB)
Full	0.9845	0.98	0.99	0.985	1845.49	9.90	1090.64
No attention	0.9758	0.97	0.98	0.975	1836.21	9.85	1087.31
No pruning	0.9832	0.98	0.99	0.984	1927.44	14.72	1328.90
No quantization	0.9839	0.98	0.99	0.984	1910.55	12.83	1295.22

Note: Values reproduce the supplied experiment record. Elapsed time is not presented as per-sample latency because the measurement interval is unspecified. Precision_w, Recall_w, and F1_w are support-weighted averages (Eqs. (5)-(7)), CPU use is given in percent, and memory in MB. Each row is a single run in which one component is removed from the full model.

Table 10. Relative effect of attention, pruning, and quantization derived from Table 9

Component Removed	Accuracy (pp)	F1_w	Time (%)	CPU (%)	Memory (%)
Attention	+0.87	+0.010	+0.5	+0.5	+0.3
Pruning	+0.13	+0.001	−4.3	−32.7	−17.9
Quantization	+0.06	+0.001	−3.4	−22.8	−15.8

4.2 Predictive performance

ClassicCNN produced the highest reported accuracy (98.57%) and weighted precision (0.99). LightCNN was slightly lower in accuracy (98.45%) but achieved the highest reported weighted recall (0.99) and F1-score (0.985), as shown in Table 11 and Figure 5. In a security setting, a recall advantage would be operationally important because missed attacks can be more costly than additional false alarms. Still, weighted recall must equal accuracy for single-label classification, so neither the reported 0.99 for LightCNN nor the reported 0.97 for ClassicCNN can be reconciled with accuracies of 98.45% and 98.57%, and the recall and F1 ranking of the two models is not established until this discrepancy is resolved. LSTM underperformed both convolutional models, achieving an accuracy of 94.60% and a weighted F1-score of 0.95. Because no per-class results were recorded, Table 12 bounds the minority-class exposure. At the reported accuracies, the entire error budget could fall on the minority classes, reducing their aggregate recall to about 26% for LightCNN and 32% for ClassicCNN. In contrast, the LSTM error budget exceeds the minority-class support.

Table 11. Predictive comparison with primary baselines

Model	Accuracy	Weighted Precision	Weighted Recall	Weighted F1-Score
LightCNN	0.9845	0.98	0.99	0.985
ClassicCNN	0.9857	0.99	0.97	0.980
LSTM	0.9460	0.95	0.95	0.950

Note: Lightweight Convolutional Neural Network (LightCNN), long short-term memory (LSTM).

Table 12. Accuracy-implied error budget compared with approximate minority-class test support

Model	Accuracy	Maximum Misclassified Test Samples	Approx. Minority-Class Test Support	Worst-Case Aggregate Minority Recall
LightCNN	0.9845	≈26,701	≈36,233	≈26.3%
ClassicCNN	0.9857	≈24,634	≈36,233	≈32.0%
LSTM	0.9460	≈93,023	≈36,233	0% (bound uninformative)

Note: Lightweight Convolutional Neural Network (LightCNN), long short-term memory (LSTM).

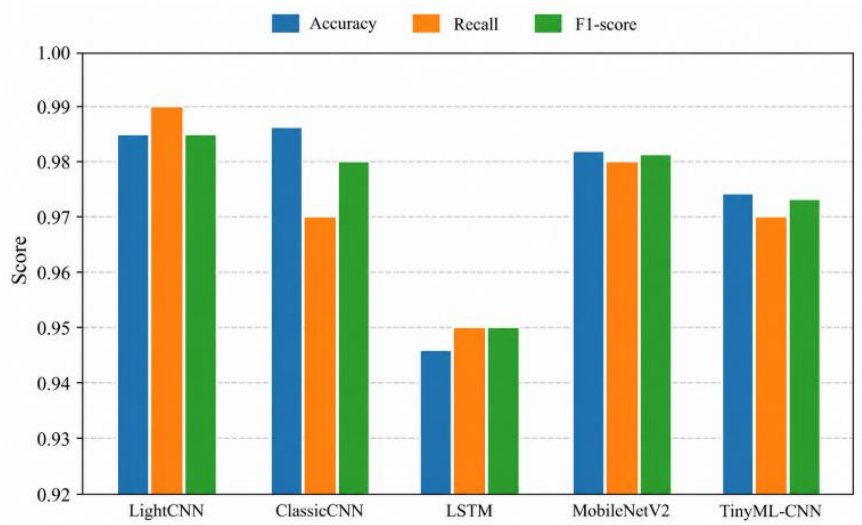


Figure 5. Predictive performance across the proposed and comparison models (author-generated visualization from the supplied results)

4.3 Computational measurements and efficiency

LightCNN had the lowest reported CPU usage and memory consumption among the three primary models (Table 13), and its elapsed time was 5.1% lower than ClassicCNN's and 21.5% lower than LSTM's. The reconstructed efficiency score is 1.49 for LightCNN, 1.26 for ClassicCNN, and 0.95 for LSTM. In this three-model set, the normalization maxima are those of LSTM, which fixes its Cost at 1.0000, and LightCNN ranks above ClassicCNN and LSTM. This result does not imply that LightCNN is universally faster; it indicates the best within-experiment balance under the specified weighting and normalization.

4.4 Comparison with lightweight architectures

MobileNetV2 achieved 98.12% accuracy and a 0.981 weighted F1-score, but its reported elapsed time, CPU use, and memory were higher than LightCNN's. TinyML-CNN had the lowest elapsed time, CPU usage, and memory among the three lightweight models, but its accuracy was 97.38%, and its F1 Score was 0.973. LightCNN, therefore, occupied the strongest reported predictive position among the three lightweight models (Table 14). Its reported efficiency score of 1.49 exceeds the reported 1.27 for MobileNetV2 and 1.29 for TinyML-CNN (Figure 6). The vertical axis plots efficiency scores that approximate the values reported in Tables 13 and 14 (Eqs. (8)-(10)), and the horizontal axis shows accuracy. However, recomputing Eqs. (8)-(10) using the normalization maxima in Table 13 yields 1.31 for MobileNetV2 and 1.57 for TinyML-CNN (Table 15), so the reported values of 1.27 and 1.29 could not be reproduced. TinyML-CNN ranks above LightCNN across all weight settings examined because it has lower elapsed time, CPU usage, and memory usage. The efficiency advantage of LightCNN over the lightweight baselines is therefore not established.

Table 13. Resource measurements and reconstructed performance-to-cost score

Model	Elapsed Time (s)	CPU (%)	Memory (MB)	Perf	Cost	Efficiency
LightCNN	1845.49	9.90	1090.64	0.9849	0.6599	1.49
ClassicCNN	1944.48	17.70	1346.62	0.9814	0.7766	1.26
LSTM	2349.81	29.00	1583.56	0.9490	1.0000	0.95

Note: Perf and Cost were recalculated using Eqs. (8) and (9). Rounding explains minor differences in the last decimal place. Cost is normalized by the maximum elapsed time (2349.81 s), CPU usage (29.00%), and memory usage (1583.56 MB) across the three models, with weights of 0.60, 0.25, and 0.15, and Efficiency is Perf divided by Cost (Eq. (10)). All values come from a single recorded run on an unspecified platform. Lightweight Convolutional Neural Network (LightCNN), long short-term memory (LSTM).

Table 14. Comparison with reported lightweight baselines

Model	Accuracy	Precision_w	Recall_w	F1_w	Time (s)	CPU (%)	Memory (MB)	Efficiency
LightCNN	0.9845	0.98	0.99	0.985	1845.49	9.90	1090.64	1.49
MobileNetV2	0.9812	0.98	0.98	0.981	2024.73	13.40	1248.55	1.27
TinyML-CNN	0.9738	0.97	0.97	0.973	1765.18	8.75	986.12	1.29

Note: Lightweight Convolutional Neural Network (LightCNN).

Table 15. Recomputed efficiency scores under alternative cost-weight settings

Weights (Time/CPU/Memory)	LightCNN	ClassicCNN	LSTM	MobileNetV2	TinyML-CNN
Reported (Tables 13 and 14)	1.49	1.26	0.95	1.27	1.29
0.60/0.25/0.15 (Eq. (9))	1.49	1.26	0.95	1.31	1.57
0.33/0.33/0.33	1.63	1.29	0.95	1.39	1.74
0.25/0.60/0.15	1.95	1.40	0.95	1.61	2.10
0.15/0.25/0.60	1.60	1.25	0.95	1.37	1.73

Note: Lightweight Convolutional Neural Network (LightCNN), long short-term memory (LSTM).

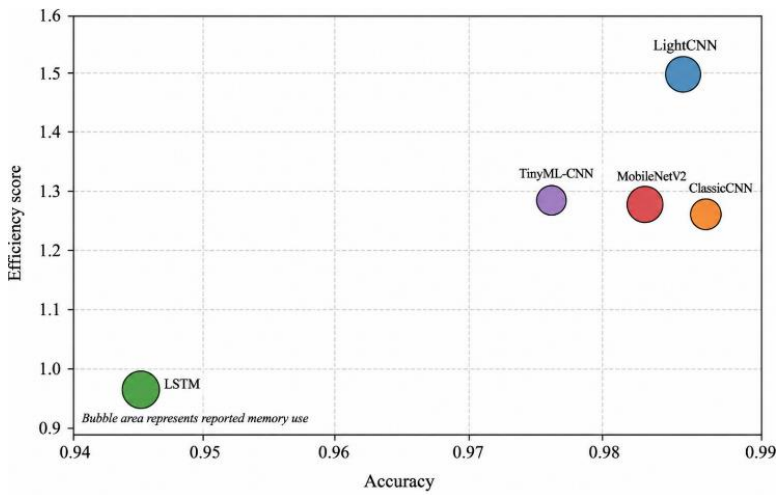


Figure 6. Accuracy-efficiency relationship; bubble area represents reported memory use (author-generated visualization from the supplied results)

5. DISCUSSION

5.1 Interpretation of the main findings

The ablation results support the study's hypothesis that attention and compression address different aspects of the deployment problem. Attention materially improved predictive balance, while pruning and quantization reduced resource demand with only small changes in reported accuracy. This pattern is consistent with recent attention-based IoT intrusion studies and with established compression research showing that redundant weights and high-precision arithmetic can often be reduced without a proportional loss of accuracy. These statements are confined to a single recorded run on an unspecified platform because repeated-run variability was not reported, and the accuracy differences of 0.06 and 0.13 percentage points associated with pruning and quantization cannot be separated from run-to-run variation.

The comparison also suggests that a marginally lower accuracy can still be acceptable when resource demand is substantially lower. ClassicCNN exceeded LightCNN by 0.12 percentage points in accuracy. In contrast, LightCNN showed substantially lower CPU and memory usage and higher reported weighted recall and F1 Score. However, the recall-based advantage depends on resolving the discrepancy between weighted recall and accuracy noted in Section 4.2. In safety- or security-critical systems, the model-selection criterion should reflect false-negative cost, device constraints, and throughput requirements rather than accuracy alone. The proposed efficiency index is one way to make that trade-off explicit. However, deployment-specific weights should be justified through operational requirements, and the ranking against TinyML-CNN in Table 15 shows that efficiency conclusions depend on the normalization set and on the weights.

5.2 Comparison with the supplied evidence synthesis

The secondary evidence report emphasizes structured pruning, quantization-aware training, hybrid attention, knowledge distillation, multimodal fusion, adversarial robustness, and explainable AI. The present experiment directly evaluates only a subset of that agenda. Knowledge distillation, multimodal late fusion, adversarial testing, and federated adaptation should not be described as completed contributions; they are directions for future research. Table 16 summarizes how each proposition is supported by the present evidence, including the assessment that real-time edge deployment is not established.

Table 16. Reconciliation of the experimental manuscript with the supplied evidence synthesis

Evidence-Synthesis Proposition	Evidence in this Experiment	Assessment
Attention-enhanced lightweight models improve discrimination	Supported by the no-attention ablation: F1 falls from 0.985 to 0.975	Strong within-experiment support
Pruning and quantization reduce deployment cost	Supported by lower CPU and memory in the full configuration	Support is empirical, but compression details are incomplete
Multi-vector and multi-source data improve robustness	The study combines several sources and scenarios	Robustness cannot be isolated without source-wise or cross-dataset tests
Explainability is required	Attention weights are available conceptually	No analyst study, SHAP, or Grad-CAM result is reported
Real-time edge deployment is feasible	Resource measurements are favorable.	Not established: hardware, per-sample latency, energy, and sustained throughput are absent
Adversarial and zero-day robustness remain open	No dedicated adversarial protocol is reported	Unresolved and appropriate for future work

5.3 Deployment implications

The reported memory footprint of approximately 1.09 GB is lower than that of ClassicCNN, LSTM, and MobileNetV2 but higher than that of TinyML-CNN, and it remains high for many microcontrollers and low-end IoT nodes. The current configuration is therefore more plausibly suited to an edge gateway, embedded Linux board, or nearby fog node than to deeply constrained sensor hardware, and the reported results describe resource optimization on an unspecified experimental platform rather than validation on a physical edge device. No inference time, power consumption, or throughput was measured on an edge platform in this study, and model file size, peak resident memory, energy per inference, and throughput must be measured on at least one intended platform before statements about real-time or edge-device operation are made. A credible edge evaluation should distinguish among preprocessing time, model invocation time, communication overhead, warm- and cold-start behavior, and batch-size effects.

Attention weights may assist interpretation by identifying influential token positions, but attention alone is not a complete explanation. Post hoc methods such as Grad-CAM for convolutional features or SHAP for feature contributions can provide complementary views. Explanations should be evaluated for stability, faithfulness, and analyst usefulness rather than presented solely as visual decoration.

5.4 Threats to validity and limitations

Incomplete hyperparameters, compression metadata, and comparison conditions limit internal validity, and weighted recall values that differ from accuracy indicate a reporting inconsistency that must be resolved. Construct validity is limited because elapsed experimental time does not equate to inference latency, and the study-specific efficiency score is sensitive to its weights and to the normalization set. The absence of repeated-run variability, confidence intervals, and significance testing limits the validity of statistical conclusions. In contrast, external validity is limited by unresolved class accounting, uncertain source overlap, the absence of a leave-one-dataset-out experiment, and the lack of direct measurement on heterogeneous edge hardware. The absence of adversarial evasion, concept drift, encrypted traffic, and true zero-day protocols limits security validity.

The severe class imbalance also requires more granular

reporting. The majority class (RCE approximately 97.9%) can dominate weighted averages, and the reported accuracies are compatible with very different minority-class recall values, so macro F1-score, class-wise precision and recall, confusion matrices, precision-recall curves, and detection rates at operational false-positive thresholds should be included in the final experiment package. Consequently, differences in detection among normal traffic, DDoS, RCE, and API exploitation or data exfiltration cannot be discussed from the present record, and no claim about minority attack classes is made. Separate sampling and dataset leakage can materially inflate performance estimates, so duplicate and group-aware splitting must be demonstrated.

6. CONCLUSIONS

The supplied single-run experiment indicates that an attention-enhanced LightCNN can achieve a favorable balance between predictive performance and measured computational demand for multi-source IoT malware detection, positioning the work as a resource-oriented optimization rather than a validated edge deployment. The full model reported an accuracy of 98.45%, a weighted recall of 0.99 (which is not fully consistent with the accuracy), and a weighted F1-score of 0.985. Removing attention produced the largest decline in predictive performance, whereas removing pruning or quantization mainly increased CPU and memory demand. LightCNN achieved the highest reconstructed efficiency score among the three primary models. Still, a recomputation under a common normalization ranked TinyML-CNN higher, so its efficiency advantage over lightweight baselines is not established, and the score depends on the chosen weights.

The principal conclusion is narrower than a claim of proven real-time deployment: the architecture is a candidate for edge-oriented validation, because no inference latency, power consumption, or throughput was measured on an edge device. Before submission and operational use, the authors should resolve class accounting, release preprocessing and mapping artifacts, specify all hyperparameters and compression procedures, report parameters and model size, repeat experiments with uncertainty estimates, add per-class analyses, and benchmark true inference latency and energy on target hardware. Until such measurements and per-class analyses are available, the results should be interpreted as evidence of resource optimization on an unspecified platform, and the efficiency ranking should be treated as study-specific. With these additions, the framework can provide a rigorous computational measurement contribution to efficient IoT security.

AUTHOR CONTRIBUTIONS

The authors contributed to this study in various significant ways. Andi Novianto conceptualized the research, led the design and methodology, and supervised the entire project, overseeing data collection, analysis, and manuscript drafting. Fatchul Arifin assisted with the literature review and data analysis, refined the research framework, and contributed to the manuscript's writing and editing. Herman Dwi Surjono played a crucial role in setting up the experiments and implementing the machine learning models, while also providing insights during the performance evaluation. Mochamad Subchan Mauludin focused on extensive data preprocessing and the development of the model's attention mechanism, contributing to the methods section and providing essential technical support. Lastly, Singgih Dwi Prasetyo was involved in validating the results, assessing the model's practical applicability, and offering critical feedback on the final manuscript. All authors approved the final version of the manuscript and agreed to take responsibility for all aspects of the work.

ACKNOWLEDGMENT

This work is part of the doctoral research conducted in the Department of Engineering Science at Universitas Negeri Yogyakarta.

REFERENCES

[1] Chataut, R., Phoummalayvane, A., Akl, R. (2023). Unleashing the power of IoT: A comprehensive review of IoT applications and future prospects in healthcare, agriculture, smart homes, smart cities, and industry 4.0. Sensors, 23(16): 7194. https://doi.org/10.3390/s23167194

[2] Al-Sinayyid, A., Sasidhar, K., Ali Jewel, M.J., MAnnuru, V. (2023). A literature survey and analysis of defending cyber attacks targeting IoT in critical infrastructure. In 2023 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, pp. 823-829. https://doi.org/10.1109/CSCI62032.2023.00139

[3] Laghari, S.U.A., Li, W.H., Manickam, S., Nanda, P., Al-Ani, A.K., Karuppayah, S. (2024). Securing MQTT ecosystem: Exploring vulnerabilities, mitigations, and future trajectories. IEEE Access, 12: 139273-139289. https://doi.org/10.1109/ACCESS.2024.3412030

[4] Al-Garadi, M.A., Mohamed, A., Al-Ali, A.K., Du, X.J., Ali, I., Guizani, M. (2020). A survey of machine and deep learning methods for Internet of Things (IoT) security. IEEE Communications Surveys & Tutorials, 22(3): 1646-1685. https://doi.org/10.1109/COMST.2020.2988293

[5] Islam, M.M., Salsabil, U., Nurmamatov, M., Hossain, S. (2026). Lightweight intrusion detection systems for IoT-edge environments: A PRISMA-ScR systematic review of deployability evidence and a unified assessment framework. Future Internet, 18(6): 300. https://doi.org/10.3390/fi18060300

[6] Liang, T.L., Glossner, J., Wang, L., Shi, S.B., Zhang, X.T. (2021). Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing, 461: 370-403. https://doi.org/10.1016/j.neucom.2021.07.045

[7] Sabir, B., Ullah, F., Babar, M.A., Gaire, R. (2022). Machine learning for detecting data exfiltration. ACM Computing Surveys, 54(3): 1-47. https://doi.org/10.1145/3442181

[8] Catalano, C., Chezzi, A., Angelelli, M., Tommasi, F. (2022). Deceiving AI-based malware detection through polymorphic attacks. Computers in Industry, 143: 103751. https://doi.org/10.1016/j.compind.2022.103751

[9] Akhi, M., Eising, C., Dhirani, L.L. (2026). Securing IoT using lightweight TCN for edge deployment on raspberry

Pi 4. IEEE Open Journal of the Communications Society, 7: 442-460. https://doi.org/10.1109/OJCOMS.2025.3649498
[10] Misrak, S.F., Melaku, H.M. (2025). Lightweight intrusion detection system for IoT with improved feature engineering and advanced dynamic quantization. Discover Internet of Things, 5(1): 97. https://doi.org/10.1007/s43926-025-00203-8
[11] Bakhsh, S.A., Khan, M.A., Ahmed, F., Alshehri, M.S., Ali, H., Ahmad, J. (2023). Enhancing IoT network security through deep learning-powered intrusion detection system. Internet of Things, 24: 100936. https://doi.org/10.1016/j.iot.2023.100936
[12] Tseng, S.M., Wang, Y.Q., Wang, Y.C. (2024). Multi-class intrusion detection based on transformer for IoT networks using CIC-IoT-2023 dataset. Future Internet, 16(8): 284. https://doi.org/10.3390/fi16080284
[13] Trisnoaji, Y., Prasetyo, S.D., Arifin, Z., Mauludin, M.S. (2026). Enhancing sustainability of on-grid hybrid PV-wind systems for green hydrogen production from unconventional resources: A meta-analytical and deep learning approach. Fuel, 422: 139157. https://doi.org/10.1016/j.fuel.2026.139157
[14] Prasetyo, S.D., Trisnoaji, Y., Kusumawardana, A., et al. (2026). Advancing renewable energy thermal management with cellulose-derived materials through meta-analysis, machine learning, and comparative material evaluation. Next Materials, 12: 102557. https://doi.org/10.1016/j.nxmate.2026.102557
[15] Abdalgawad, N., Sajun, A., Kaddoura, Y., Zualkernan, I.A., Aloul, F. (2022). Generative deep learning to detect cyberattacks for the IoT-23 dataset. IEEE Access, 10: 6430-6441. https://doi.org/10.1109/ACCESS.2021.3140015
[16] Owoh, N., Adejoh, J., Hosseinzadeh, S., Ashawa, M., Osamor, J., Qureshi, A. (2024). Malware detection based on API call sequence analysis: A gated recurrent unit-generative adversarial network model approach. Future Internet, 16(10): 369. https://doi.org/10.3390/fi16100369
[17] C, A., R, V., M, M., R, A.C. (2025). TinyML-based intrusion detection systems for sustainable and energy-constrained IoT devices. Results in Engineering, 28: 108013. https://doi.org/10.1016/j.rineng.2025.108013
[18] Dutta, D.L., Bharali, S. (2021). TinyML meets IoT: A comprehensive survey. Internet of Things, 16: 100461. https://doi.org/10.1016/j.iot.2021.100461
[19] Priyadarshini, I., Mohanty, P., Alkhayyat, A., Sharma, R., Kumar, S. (2023). SDN and application layer DDoS attacks detection in IoT devices by attention-based Bi-LSTM-CNN. Transactions on Emerging Telecommunications Technologies, 34(11): e4758. https://doi.org/10.1002/ett.4758
[20] Altunay, H.C., Albayrak, Z. (2023). A hybrid CNN+LSTM-based intrusion detection system for industrial IoT networks. Engineering Science and Technology, an International Journal, 38: 101322. https://doi.org/10.1016/j.jestch.2022.101322
[21] Li, Z., Li, H.Y., Meng, L. (2023). Model compression for deep neural networks: A survey. Computers, 12(3): 60. https://doi.org/10.3390/computers12030060
[22] Schaefer, C.J.S., Joshi, S., Li, S., Blazquez, R. (2024). Edge inference with fully differentiable quantized mixed precision neural networks. In 2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), Waikoloa, HI, USA, pp. 8445-8454. https://doi.org/10.1109/WACV57701.2024.00827
[23] Guo, Y.H., Du, C.L., Mustafaoglu, Z., et al. (2025). Using hybrid transformer and convolutional neural network for malware detection in Internet of Things. International Journal of Pattern Recognition and Artificial Intelligence, 39(02): 2550002. https://doi.org/10.1142/S0218001425500028
[24] Sani Iliyasu, A., Jabbar Siddiqui, A., Song, H.B., Jibrin Abdu, F. (2025). PNet-IDS: A lightweight and generalizable convolutional neural network for intrusion detection in Internet of Things. IEEE Access, 13: 102624-102639. https://doi.org/10.1109/ACCESS.2025.3575705
[25] Cao, J.J., Yu, C., Ye, P., Chen, T. (2026). MADTP++: Bridge the gap between token and weight pruning for accelerating VLTs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 48(5): 5180-5194. https://doi.org/10.1109/TPAMI.2025.3650545
[26] Xiao, P.H., Zhang, C.J., Guo, Q., Xiao, X.Y., Wang, HP. (2024). Neural networks integer computation: Quantizing convolutional neural networks of inference and training for object detection in embedded systems. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 17: 15862-15884. https://doi.org/10.1109/JSTARS.2024.3452321
[27] Neto, E.C.P., Dadkhah, S., Ferreira, R., Zohourian, A., Lu, R.X., Ghorbani, A.A. (2023). CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment. Sensors, 23(13): 5941. https://doi.org/10.3390/s23135941
[28] Shahrokh Esfahani, M., Dougherty, E.R. (2014). Effect of separate sampling on classification accuracy. Bioinformatics, 30(2): 242-250. https://doi.org/10.1093/bioinformatics/btt662
[29] Kim, S., Gholami, A., Yao, Z.W., et al. (2022). Integer-only zero-shot quantization for efficient speech recognition. In ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Singapore, pp. 4288-4292. https://doi.org/10.1109/ICASSP43922.2022.9747552
[30] Yin, R.K., Li, Y.H., Moitra, A., Panda, P. (2024). MINT: Multiplier-less INTeger quantization for energy efficient spiking neural networks. In 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC), Incheon, Republic of Korea, pp. 830-835. https://doi.org/10.1109/ASP-DAC58780.2024.10473825
[31] Prasetyo, S.D., Arifin, Z., Trisnoaji, Y., Rosli, M.A.M., Fudholi, A. (2026). Advanced hybrid cooling approach to improve thermal management in photovoltaic thermal collectors utilizing nanofluids and nano-enhanced bio-phase change materials. Next Materials, 13: 103545. https://doi.org/10.1016/j.nxmate.2026.103545
[32] Ahmad, B., Li, Y.M., Wu, Z.L., Rehman, S.U., Huang, Y.F. (2026). Improved attack detection in IoT and IIoT networks using attention mechanisms in convolutional neural networks. Expert Systems with Applications, 296: 129021. https://doi.org/10.1016/j.eswa.2025.129021
[33] Krishnan, D., Singh, S., Sugumaran, V. (2025). Explainable AI for zero-day attack detection in IoT networks using attention fusion model. Discover Internet of Things, 5(1): 83. https://doi.org/10.1007/s43926-025-

00184-8
[34] Li, Z.K., Gu, Q.Y. (2023). I-ViT: Integer-only quantization for efficient vision transformer inference. In 2023 IEEE/CVF International Conference on Computer Vision (ICCV), Paris, France, pp. 17019-17029. https://doi.org/10.1109/ICCV51070.2023.01565
[35] Nia, V.P., Sari, E., Courville, V., Asgharian, M. (2023). Training integer-only deep recurrent neural networks. SN Computer Science, 4(5): 501. https://doi.org/10.1007/s42979-023-01920-z
[36] Alandjani, G. (2024). Optimizing malware detection for IoT and edge environments with quantization awareness. IEEE Access, 12: 166776-166791. https://doi.org/10.1109/ACCESS.2024.3495635
[37] Wang, M.L., Rasoulinezhad, S., Leong, P.H.W., So, H.K.H. (2022). NITI: Training integer neural networks using integer-only arithmetic. IEEE Transactions on Parallel and Distributed Systems, 33(11): 3249-3261. https://doi.org/10.1109/TPDS.2022.3149787
[38] Maddiralla, V., Subramanian, S. (2024). Effective lane detection on complex roads with convolutional attention mechanism in autonomous vehicles. Scientific Reports, 14(1): 19193. https://doi.org/10.1038/s41598-024-70116-z
[39] Guo, M.H., Xu, T.X., Liu, J.J., et al. (2022). Attention mechanisms in computer vision: A survey. Computational Visual Media, 8(3): 331-368. https://doi.org/10.1007/s41095-022-0271-y
[40] Zhang, Q.X., Kanjo, E. (2026). A multicore and edge TPU-accelerated multimodal TinyML system for livestock behavior recognition. IEEE Internet of Things Journal, 13(1): 666-677. https://doi.org/10.1109/JIOT.2025.3624811
[41] Mauludin, M.S., Rifan, A., Prasetyo, S.D., Trisnoaji, Y. (2026). A hybrid framework integrating meta-analytic evidence, physics-informed deep learning, and monte carlo simulation for risk and optimization of green hydrogen production via hybrid renewable systems. Semarak Engineering Journal, 15(1): 125-148. https://doi.org/10.37934/sej.15.1.125148

APPENDIX

Appendix A. Mandatory pre-submission verification

Table A1. Verification checklist before journal submission

Area	Required Item	Current Status	Action
Author metadata	Names, affiliations, institutional emails, ORCID, corresponding author	Not supplied	Replace highlighted placeholders
Class accounting	Mutually exclusive counts for all final labels	Inconsistent	Regenerate from the preprocessing pipeline
Data provenance	DOI/URL, version, license, checksum for every source and derived file	Partial	Add repository and data dictionary
Split integrity	Deduplication, source grouping, leakage prevention	Not documented	Release split IDs and audit log
Architecture	Filters, kernels, pooling, dropout, attention dimension	Incomplete	Export model summary/configuration
Training	Optimizer, learning rate, batch size, epochs, seed, early stopping	Incomplete	Add exact settings and code
Compression	Pruning type/schedule/sparsity; QAT or PTQ; calibration	Incomplete	Add full conversion protocol
Complexity	Parameter count, MACs/FLOPs, model file size	Missing	Measure before and after compression
Hardware measurements	Device, OS, framework, threads, warm-up, batch size	Missing	Repeat on named target hardware
Statistics	Number of runs, mean, SD/CI, significance test	Missing	Repeat experiments with fixed seeds
Class-wise evaluation	Confusion matrix, macro metrics, PR curves	Missing	Add all classes and operational thresholds
Repository	Code, data-processing scripts, checkpoints, logs	Missing	Deposit with a persistent identifier

Note: The manuscript should not be submitted until all highlighted placeholders and the authors' addresses unresolved items.

Table A2. Processing pipeline of the attention-enhanced LightCNN (F, k, and d_a denote unreported hyperparameters)

1	Input: labeled records from CICIoT2023, IoT-23, Infiltration.csv, and the controlled API simulation; class set of four labels.
2	Harmonize source labels to the four classes and convert each record to attribute-value tokens, replacing numerical, hexadecimal, and variable values by <NUM>, <HEX>, and <PARAM>.
3	Build a vocabulary of at most 10,000 tokens with padding and unknown markers; pad or truncate each sequence to T = 128 token identifiers.
4	Split the data by stratified sampling (seed 42) into 70% training, 15% validation, and 15% testing partitions.
5	Embed the tokens (d = 128) and compute local features H = ReLU(Conv1D(E; F, k)) [Eq. (1)].
6	For each position t, compute $u_t = \tanh(W_a h_t + b_a)$ and $\alpha_t = \text{softmax over t of } v^T u_t$, then form $c = \Sigma t\ \alpha_t h_t$ [Eqs. (2) and (3)].
7	Compute $p(y \mid x) = \text{softmax}(W_o c + b_o)$ [Eq. (3)].
8	Train by minimizing categorical cross-entropy [Eq. (4)] with the Adam optimizer suggested in the source pseudocode; learning rate, batch size, epochs, and early stopping are unreported.

9	Prune the network weights (types, ratios, and schedules unreported) and convert the pruned model to INT8 (post-training or quantization-aware conversion unreported).
10	Evaluate accuracy and weighted precision, recall, and F1-score [Eqs. (5)-(7)] on the test partition and record elapsed time, CPU use, and memory use [Eqs. (8)-(10)].
11	Output: compressed classifier, predictive metrics, and resource measurements.