ISNNI: 3007-3391 HEC Category: Y

Journal of Computers and Intelligent Systems
Vol. 4, No. 3, 2026, pages 407 - 419
Journal Homepage:
https://journals.iub.edu.pk/index.php/JCIS/

Graph Neural Networks for Multiclass IoT Botnet Detection: A Comparative Study of In-Distribution and Cross-Device Generalization Using N-BaIoT

Nadeem Ahmed[1], Fayaz Hassan[1], Shakeel Laghari[1], Ahmed Asghar[2], Ayesha Eman[1]
[1] Department of Telecommunication Engineering, Mehran University of Engineering and Technology, Jamshoro, Pakistan
[2] Department of Computer Science and Related Studies (CSRS), Hyderabad Institute for Technology and Management Sciences, Hyderabad, Pakistan

ARTICLE INFO

Article History:

Received:	July	11, 2026
Revised:	August	21, 2026
Accepted:	September	26, 2026
Available Online:	September	30, 2026

Keywords:

Botnet Detection
Cross-Device Generalization
Graph Neural Networks
IoT Security
N-BaIoT
Keyword 5

Classification Codes:

Funding:

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

ABSTRACT

The widespread deployment of IoT devices across critical infrastructure has significantly widened the attack surface for botnet-based intrusions, with Mirai and BASHLITE among the most destructive documented threats. Despite extensive research on IoT intrusion detection, most studies still rely on random train/test splits drawn from the same pool of devices, which does not directly test generalization to unseen hardware. This paper addresses that deployment-oriented question. We present a two-layer Graph Convolutional Network (GCN) for multiclass IoT botnet detection using the N-BaIoT dataset, covering 11 traffic classes across three commercial IoT devices, and evaluate it alongside Random Forest and XGBoost under two protocols: a conventional in-distribution split (Protocol 1) and a cross-device split in which all samples from the test device are excluded from training (Protocol 2). Across five seeds, Protocol 1 accuracies were 94.41 ± 0.04% for Random Forest, 99.95 ± 0.04% for XGBoost and 86.22 ± 0.39% for GCN. On the held-out Philips device, the corresponding accuracies were 99.77 ± 0.04%, 99.59 ± 0.11% and 84.06 ± 19.95%, with GCN variability reflecting one unstable run. The GCN also reproduces a Mirai ACK/UDP confusion pattern across the two protocols and on the held-out device, while Protocol 1 reveals additional confusion among BASHLITE flood subtypes. Class-level analysis and graph-sensitivity experiments identify errors and variability in the tested models that aggregate accuracy alone can obscure. The paired in-distribution and held-out-device evaluation provides a more deployment-oriented comparison of graph-based and flat classifiers.

OPEN ACCESS

© 2026 The authors published by JCIS. This is an Open Access Article under the Creative Common Attribution Non-Commercial 4.0

Corresponding Author's Email: nadeem.ahmed@admin.muet.edu.pk

1. Introduction

The Internet of Things has become a part of many areas of everyday life; nowadays, it is widely used, from smart homes and healthcare to industrial automation and critical infrastructure. Along with its advantages, as more devices become connected, securing these systems has become an important concern. Many IoT devices have limited processing power, basic built-in security, and default credentials that users often don't change. [1,2].

Intrusion Detection Systems (IDSs) are the primary technical solution for addressing this threat. Machine learning-based approaches, particularly ensemble methods such as Random Forest and gradient boosting (XGBoost), have shown high accuracy on benchmark IoT datasets [3–5], as confirmed by recent surveys of the field [6]. However, the standard evaluation methodology in much of this literature relies on random splits from a fixed device pool. Cross-device IoT botnet detection has also been investigated, including an all-in-one deep autoencoder evaluated across N-BaIoT

devices by Catillo et al. [7]. The specific gap addressed here is the controlled comparison of a GCN and conventional flat classifiers under identical in-distribution and fully held-out-device protocols, with multiclass confusion analysis under the same experimental design.

Graph Neural Networks offer a natural alternative framing for network traffic analysis. Rather than treating each flow record independently, GNNs represent flows as graph nodes and learn from the relational structure among them. Bilot et al. [8] review the use of GNNs for intrusion detection and highlight their ability to model relational structure. Practical demonstrations include E-GraphSAGE [9], E-GRACL [10], HADGA [11], and the Graph Attention Network (GAT) approach of Ahanger et al. [12]. Recent work by Tanvir et al. [13] further demonstrated spatial-temporal GNN evaluation across domains. In parallel, Catillo et al. [7] investigated cross-device IoT botnet detection on N-BaIoT. In this background, we compare GCN, Random Forest and XGBoost in a head-to-head manner on the same N-BaIoT samples for both conventional and fully held-out-device evaluation conditions.

The contribution is an empirical comparison with class-level error analysis and graph-sensitivity experiments, scoped to the selected devices and known attack classes; no new GNN architecture is proposed.

This paper makes the following specific contributions:

1. We conduct a controlled cross-device evaluation on N-BaIoT, training on two devices and testing on a fully excluded third device, while evaluating a GCN and two widely used flat classifiers under identical experimental conditions.
2. We observe the Mirai ACK/UDP confusion pattern under both evaluation protocols, including on the fully held-out Philips device. Protocol 1 also shows confusion between BASHLITE TCP/UDP subtypes; this cannot be evaluated across devices in Protocol 2 because the selected held-out Philips records contain no BASHLITE traffic.
3. We apply a paired in-distribution and cross-device evaluation framework as a more deployment-oriented benchmark for IoT intrusion detection and provide a direct comparison table positioning this work against selected prior studies.

Section 2 reviews related work. Section 3 details the methodology and mathematical formulation. Section 4 provides a detailed experimental setup. Section 5 presents and interprets results. Section 6 concludes with implications and future directions.

2. Related Work

Intrusion detection for IoT networks has been widely investigated, and in the literature, three directions can be distinguished: the first is based on engineered flow features and flat classifiers; the second is based on deep learning; and the third is based on graph-based models that leverage network topology.

The N-BaIoT dataset is consistently classified by classifiers leveraging statistical information for single flows. In this research, Hossain and his team [3] conducted a study with 6 different models and obtained 99 per cent accuracy with Random Forest and 54 per cent accuracy with SVM applied to the security camera device.Tarab [4] found that XGBoost is slightly more accurate, achieving 99.12 per cent accuracy and 0.018 milliseconds per sample inference time. In order to detect multiclass botnets, Taher et al. [14] used a hybrid feature selection method and an enhanced Harris Hawks optimization algorithm. Rawat et al. [5] confirmed that ensemble methods are still one of the best baseline methods for this dataset. To decrease the computational complexity, Wang et al. [15] proved that light-weight deep learning models such as BiLSTM can be as accurate as more complex models. Combined these studies mostly re-port conventional within-dataset evaluations, which do not address the fully held-out-device test in which all samples from the test hardware are not included in the training set, and thus do not directly answer the cross-device question that was explored in Protocol 2.

Deep learning surveys confirm the maturity and wide scope of this area. Recent surveys indicate that there are still difficulties in assessing the generalizability of IoT intrusion detection systems across diverse environments [6,8]. Bilot et al. [8] specifically examine intrusion detection using GNNs and emphasize the need for graph-based representations to capture structural attack patterns.

Graph Neural Networks have emerged as a prominent direction for network intrusion detection. Kipf and Welling [16] introduced the GCN layer-wise propagation rule used in this work, originally demonstrated on citation network classification. Lo et al. [9] built E-GraphSAGE, demonstrating GNNs outperforming standard classifiers across four benchmark IoT datasets. Lin et al. [10] extended this with E-GRACL, adding global attention, local gating, and contrastive

learning. HADGA is a hierarchical attention-based dynamic graph neural network proposed by Sun et al. [11] that achieves 97.6% and 99.9% accuracy in BoT-IoT and TON-IoT datasets, respectively. Villegas-Ch. et al. [17] have used dynamic graph models in IoT traffic and achieved an F1-score of 0.95 and 40% false positive reduction over signature-based IDS. Ahanger et al. [12] showed that GAT-based IDS achieves high accuracy on the NSL-KDD dataset and emphasized GAT's capability to assign different weights to neighboring nodes. Bondar [18] examined the effectiveness of GCN, GraphSAGE, and GAT models relative to Random Forest on the NSL-KDD and BoT-IoT datasets, indicating that GAT achieves around 98.3% accuracy, which is 7% higher than a CNN model. While Altaf et al. [19] studied sequential attack detection in IoT via GNNs, Yin et al. [20] proposed GraphSAINT subgraph sampling technique for large-scale botnet detection. Yang et al. [21] tackled heterogeneity issue of Industrial IoT devices using GNNs via mini-batch sampling and temporal correlation. In federated setup, Al Tfaily et al. [22] developed FedGATSage framework, which combines a client-side GAT model and GraphSAGE model on the server side, with 85% reduction in communication cost. Concejal Munoz et al. [23] proposed communication topology graph approach that showed very good performance on practical botnet dataset. Babenko et al. [24] used GNN-LSTM hybrid architecture on N-BaIoT dataset to achieve 99.88% accuracy and Brier score of 0.0015.

One of the most relevant works is by Tanvir et al. [13], who used a spatio-temporal GNN with pretraining via autoencoders on IoT healthcare data, achieving cross-domain evaluation results and obtaining a 0.942 F1-score when testing on TON-IoT dataset after training on an alternative dataset. Among the surveyed studies, Tanvir et al. [13] is one of the few recent works on GNN-based IDS that evaluate cross-domain generalization. This work continues this line of research by conducting cross-device evaluation on N-BaIoT dataset and comparing the performance of GNNs with flat classifiers in both standard and cross-device settings, using the same datasets and data splits.

Table 1: Positioning of this work relative to selected prior studies.

Study	Dataset	Multiclass	Cross-Device / Cross-Domain Evaluation	GNN vs. Flat
Hossain et al. [3]	N-BaIoT	No	No	Flat only
Tarab [4]	N-BaIoT	No	No	Flat only
Rawat et al. [5]	N-BaIoT	Yes	No	Flat only
Babenko et al. [24]	N-BaIoT	No (binary)	No	Hybrid GNN–LSTM
Tanvir et al. [13]	Healthcare IoT	Yes	Yes (cross-domain)	GNN only
Bondar [18]	NSL-KDD, BoT-Io	Yes	No	GNN vs. Flat
Catillo et al. [7]	N-BaIoT	No (anomaly detection)	Yes (cross-device)	DL only
This paper	N-BaIoT	Yes	Yes (cross-device)	GNN vs. Flat

3. Methodology

A. System Model / Framework Overview

The proposed framework includes four consecutive phases which include data ingestion from the N-BaIoT dataset, building a k-nearest neighbors similarity graph, multiclass node classification using GCN, and comparative evaluation with flat classifiers under two evaluation settings. Figure 1 is a visual depiction of the process.

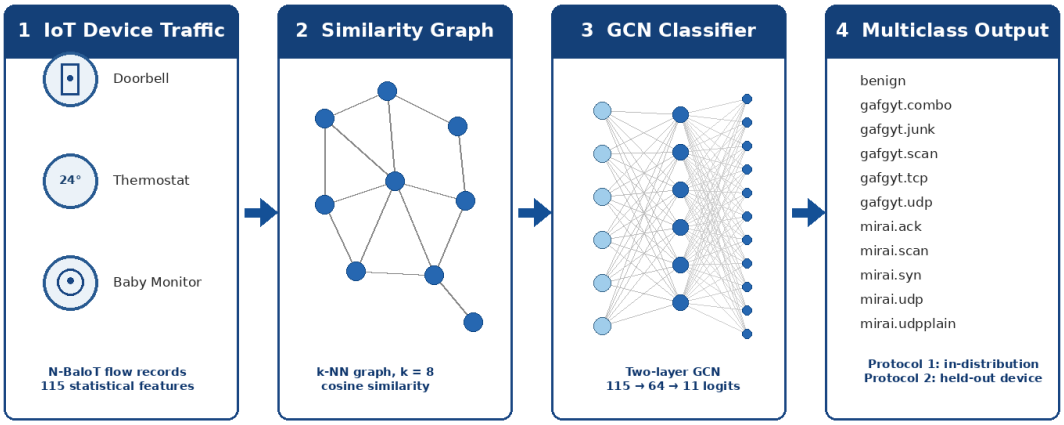


Figure 1: Overview of our system, from IoT traffic to graph construction, GCN classification, and evaluation versus Random Forest and XGBoost. Graph construction is performed independently within each partition after training-only preprocessing.

B. Dataset

We use the N-BaIoT dataset [1], which was constructed by capturing traffic from nine commercial IoT devices infected with Mirai and BASHLITE under controlled laboratory conditions. Each traffic record contains 115 statistical features computed over multiple time windows, encoding packet size distributions, inter-arrival timing, and connection counts. We choose three devices representing different product categories. The selected files consist of both benign data and Mirai transcripts for the Danmini Doorbell and Philips Baby Monitor, as well as benign data and BASHLITE transcripts for the Ecobee Thermostat. This describes the files used in this work and must not be taken as an exhaustive description of infection coverage in the original dataset. Each of the attack types is kept as a separate class to reflect the original multiclass classification of the dataset [1], resulting in 11 classes in total: benign traffic and five subtypes of Mirai (ack, scan, syn, udp, udpplain) and five subtypes of BASHLITE (combo, junk, scan, tcp, udp). Sampling of 1,200 rows per class per device was conducted, leading to a total sample size of 21,600. A summary of the dataset composition is presented in Table 2. After removing 67 duplicate features and retaining only the first occurrence in the original order, 21,533 observations remain: 7,193 from Danmini, 7,189 from Ecobee, and 7,151 from Philips. The three-device sample set provides a limited basis for comparison rather than a full representation of all devices. The number of classes and devices retained is described in Supplement S1.

Table 2: Composition of the initial sampled dataset before exact-duplicate removal.

Class	Devices Contributing	Sample Count
benign	All three devices	3,600
mirai. ack/scan / syn / udp / udpplain	Danmini Doorbell + Philips Baby Monitor	2,400 each
gafgyt. combo/junk / scan / tcp / udp	Ecobee Thermostat	1,200 each
Total	3 devices	21,600

C. Graph Construction

The traffic data is modeled as graph G = (V, E, X), where V is the set of nodes (one node per traffic flow), E is the set of edges, and X is the feature matrix for all nodes. Node v_i is paired with a feature vector x_i, which consists of the 115-dimensional statistical feature vector of the particular traffic flow.

Edges are formed using cosine similarity between node feature vectors. For two nodes v_i and v_j with feature vectors x_i and x_j:

$$\mathrm{sim}(x_i, x_j) = \frac{x_i^T x_j}{\|x_i\|_2 \|x_j\|_2} \quad (1)$$

where ||.|| denotes the L2 norm. The reference graph makes use of cosine k-nearest neighbors with k = 8. The partitions are set beforehand, and the StandardScaler is trained only on the training data. Separate training, validation, and test graphs are created without any edges or message passing between partitions. Self-neighbors are removed by node identity; unweighted edges are symmetrized and duplicate edges removed. Sensitivity experiments use cosine k=4 and 16 and Euclidean k=8. Test inference uses a batch of unlabeled test features. Communication-topology graphs require metadata unavailable in the supplied aggregate-feature sample.

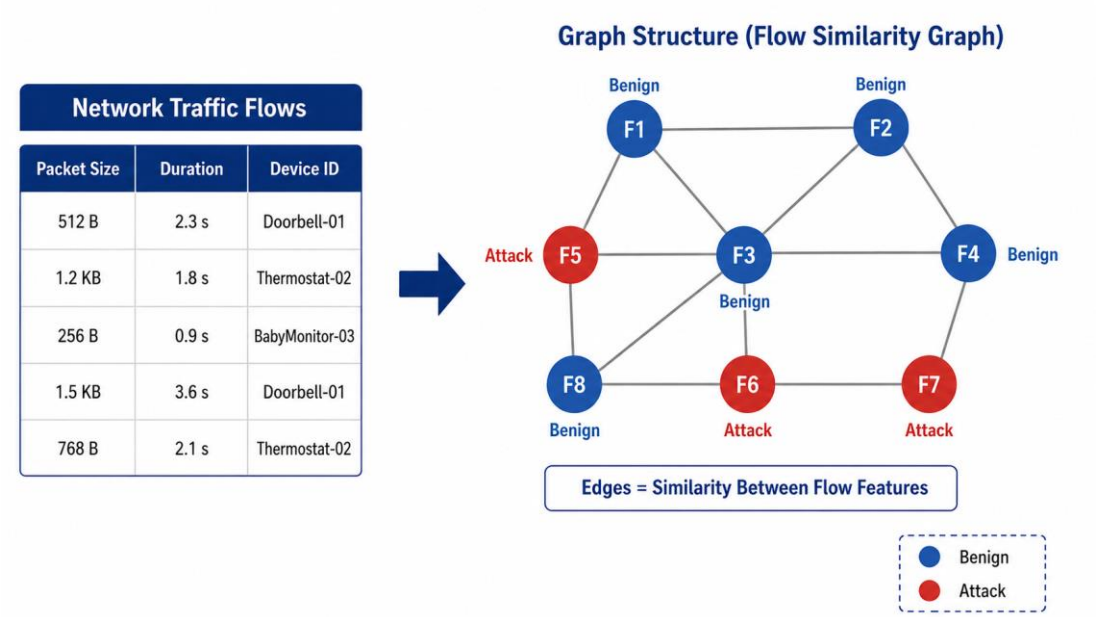


Figure 2: Illustration of the graph construction process. Raw per-flow feature records (left) are mapped to graph nodes, with edges connecting flows of high cosine feature similarity. Blue nodes represent benign flows; red nodes represent attack-class flows. This schematic illustrates within-partition edges; no cross-partition connections are used.

D. GCN Model Architecture

We implement a two-layer Graph Convolutional Network following the propagation rule of Kipf and Welling [16]. For a graph with adjacency matrix A and node feature matrix X, the normalized GCN layer-wise update rule is:

$$H^{(\ell+1)} = \sigma\left(SH^{(\ell)}W^{(\ell)} + b^{(\ell)}\right) \quad (2)$$

The self-loop adjacency, diagonal degree matrix and normalized propagation matrix S are defined below. Each partition has its own adjacency and node count. $H^{(0)} = X$, $W^{(j)}$ and $b^{(j)}$ are trainable weights and biases. Activation function used in the hidden layer is σ ReLU (Rectified Linear Unit), while the final layer produces logits without an activation function. Bias terms are broadcast to the units.

$$\tilde{A} = A + I_N, \quad \tilde{D}_{ii} = \sum_j \tilde{A}_{ij}, \quad S = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$$

$$H^{(1)} = \mathrm{ReLU}\left(SXW^{(0)} + b^{(0)}\right) \quad (3)$$

A dropout layer with probability p = 0.3 is used on $H^{(1)}$to prevent overfitting. The second layer maps the 64-dimensional representation to 11 output class logits:

$$Z = S\mathrm{Dropout}\left(H^{(1)}, 0.3\right)W^{(1)} + b^{(1)} \quad (4)$$

Final probability for each class is obtained by applying the Softmax function on the logits output Z:

$$\hat{y}_{ic} = \frac{\exp(Z_{ic})}{\sum_{r=1}^{C}\exp(Z_{ir})} \quad (5)$$

Model optimization is achieved through minimizing cross entropy loss over all the labeled training nodes:

$$\mathcal{L} = -\frac{1}{N_{\text{train}}}\sum_{i\in V_{\text{train}}}\sum_{c=1}^{C} y_{ic}\log\hat{y}_{ic} \quad (6)$$

where y_ic is the one-hot ground-truth label, ŷ_ic is the predicted probability, C=11, and N_train is the number of training nodes. Checkpoint selection uses validation cross-entropy.

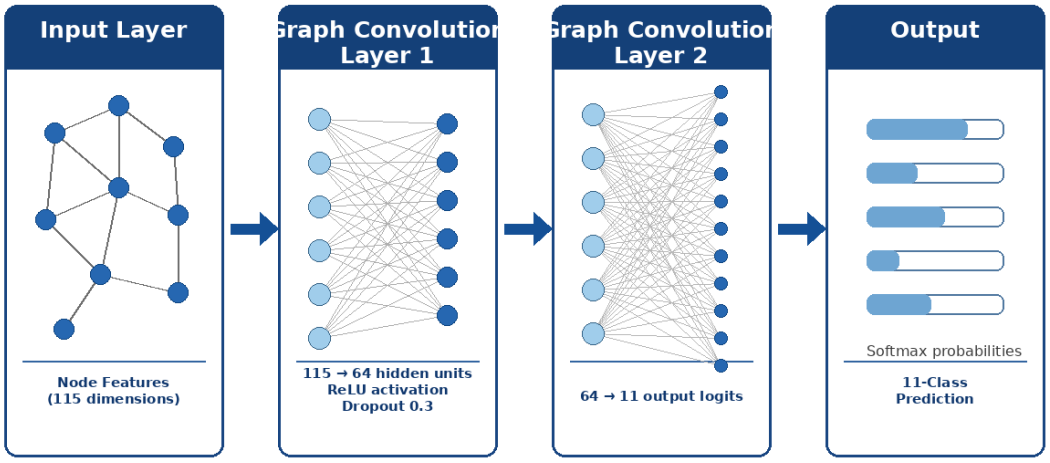


Figure 3: Two-layer GCN architecture. Node features of dimension 115 are transformed through two graph convolution layers (Equations 3 and 4) to produce 11-class output logits, with softmax applied for final classification.

E. Baseline Classifiers

We train Random Forest (100 trees) and XGBoost (100 estimators) on the same 115-dimensional feature matrix without edge information. Every record is individually classified. Additional comparators are a two-layer MLP and mean-aggregation GraphSAGE, using 64 hidden units, ReLU, dropout 0.3 and 11 outputs. All models have the training and test rows; validation records are not included in tree-model training. This makes them easier to compare. Does not separate graph aggregation as the only reason, for performance differences.

F. Evaluation Protocols

Two evaluation protocols are applied to all models. Protocol 1 (In-Distribution): Stratified 70/15/15 train/validation/test splits were used on datasets collected from all three devices and correspond to the traditional way of evaluating models inside the dataset, which is used by previous studies on N-BaIoT [3–5]. Protocol 2 (Cross-Device): Training on Danmini Doorbell and Ecobee Thermostat (14,382 retained samples with 15% held out for validation) and testing on Philips Baby Monitor (7,151 retained samples) only, where Philips is not included in training. Since Danmini and Philips devices have similar Mirai attack subtypes, it is an experiment that tests how well the attack behavior on the different hardware with possibly different network statistics can be recognized. Thus, it is a closed-set cross-device experiment, where the test class exists in training data, but the test device does not. The selected Philips test records contain benign and five Mirai classes, with no BASHLITE.

G. Evaluation Metrics

Accuracy, weighted precision, recall, and F1 score of the two protocols are given, along with their class-wise results and confusion matrix. Weighted F1 Score has been used due to imbalanced classes:

$$F1_{\text{weighted}} = \sum_c \frac{\text{support}_c}{N} \cdot F1_c \quad (7)$$

where support□ is the number of test instances whose true label is class c, N is the total number of test instances, and F1□ = 2(P□ × R□) / (P□ + R□).

4. Experimental Setup

A. Environment

Experiments were conducted on Windows with Python 3.12.10, PyTorch 2.13.0, PyTorch Geometric 2.8.0.post1, scikit-learn 1.9.0, XGBoost 3.3.0 and pandas 3.0.5. Training used CPU execution with four requested threads. The full software manifest is supplied in Supplement S1.

B. Data Split

Protocol 1 uses a stratified split of the 21,533 records selected in the ratios of 70/15/15, providing 15,073 records for training, 3,230 records for validation, and 3,230 records for testing. In Protocol 2, the 14,382 records of Danmini/Ecobee dataset are divided into 12,224 records for training and 2,158 records for validation, whereas the entire 7,151 records in Philips dataset are used for testing.

C. Sample Size and Class Imbalance

The initial sample is made up of 21,600 entries in 11 different categories, of which there are 3,600 for benign samples, 2,400 for each category of the Mirai samples and 1,200 for each category of BASHLITE samples. The figures correspond to the selected device/category files. After eliminating exact duplicates, 21,533 entries are left.

D. Hyperparameter Configuration

Table 3 shows the architecture of Graph Convolutional Network (GCN). All architectures use seeds from 42 to 46.. Random Forest uses 100 trees, Gini criterion, sqrt feature selection, unrestricted depth, min_samples_split=2, min_samples_leaf=1 and bootstrap sampling. XGBoost uses 100 estimators, multi:softprob and mlogloss, with remaining XGBoost 3.3.0 defaults. Neural comparators use the same optimizer settings as GCN, zero weight decay and restoration of the lowest-validation-loss checkpoint. No hyperparameter search was performed; saved configurations are included in Supplement S1.

Results are mean ± sample SD across five seeds. Exploratory paired sign-flip tests on seed-level accuracy differences, with Holm correction across six comparisons, returned exact two-sided p=0.0625 and adjusted p=0.375. For five pairs, the lowest possible two-sided p-value is 0.0625, and no assertion of statistically superior results is claimed. Conditional confidence intervals and tests are available in Supplement S1. Repeated seeds evaluate the variation in runs and not device replication.

Table 3: GCN hyperparameter configuration.

Hyperparameter	Value	Justification
Input features	115	All N-BaIoT statistical features
Hidden dimension	64	Standard for GCN on medium-scale graphs
Output classes	11	Benign + 5 Mirai + 5 BASHLITE subtypes
Dropout rate	0.3	Regularization to prevent overfitting
Learning rate	0.01	Fixed learning rate for all neural models
Optimizer	Adam	Widely used for GCN training
Max epochs	100	Fixed training budget; best validation checkpoint retained
Early stopping patience	10	Stops on non-improving validation loss
k (kNN edges)	8	Reference graph; k=4 and 16 also evaluated
Random seeds	42, 43, 44, 45, 46	Repeated-run reporting

5. Results and Discussion

We report results under both evaluation protocols as mean ± sample SD across five seeds. Precision, recall and F1 are weighted averages. Confusion matrices show results, for seed 42. All runs are included in the aggregate statistics.

A. In-Distribution Performance

Under Protocol 1, with training and testing drawn from all three devices, Random Forest and XGBoost substantially outperform the GCN, consistent with prior N-BaIoT literature [3–5]. Table 4 reports the results.

Table 4: In-distribution performance, mean plus or minus sample deviation, across five seeds (%). Precision, recall and F1 are weighted averages.

Model	Accuracy	Precision	Recall	F1-Score
GCN (cosine k=8)	86.22 ± 0.39	84.63 ± 1.38	86.22 ± 0.39	84.57 ± 0.59

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	94.41 ± 0.04	92.73 ± 2.47	94.41 ± 0.04	92.56 ± 0.04
XGBoost	99.95 ± 0.04	99.95 ± 0.04	99.95 ± 0.04	99.95 ± 0.04
MLP	87.91 ± 0.36	85.98 ± 0.31	87.91 ± 0.36	86.04 ± 0.35
GraphSAGE (k=8)	87.29 ± 0.40	85.50 ± 0.52	87.29 ± 0.40	85.38 ± 0.44

XGBoost showed the mean accuracy in Protocol 1. The tree-based classifiers did better than the networks tested on this data. Neighborhood mixing might be one reason for the errors in GCN models. These results do not prove that the way graphs are built is the only cause of the difference. Section 5F includes experiments that look at how sensitive the model's, to the target graph.

B. Training Behavior

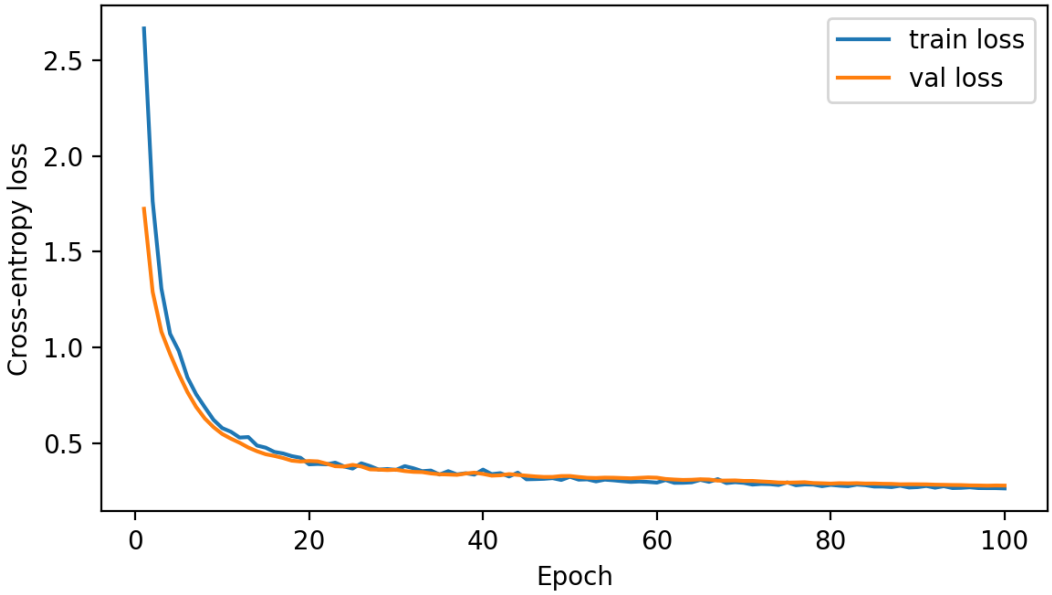


Figure 4: CNN Training and Validation Loss under Protocol 1 (Seed 42). The graph shows the loss for the particular run that supplements the aggregated result from five different seeds.

The training loss decreased during the run that is illustrated. The selection of checkpoints was done based on validation loss. Nonetheless, these plots alone cannot set a ceiling on representation or lack optimization sensitivity.

C. In-Distribution Confusion Analysis

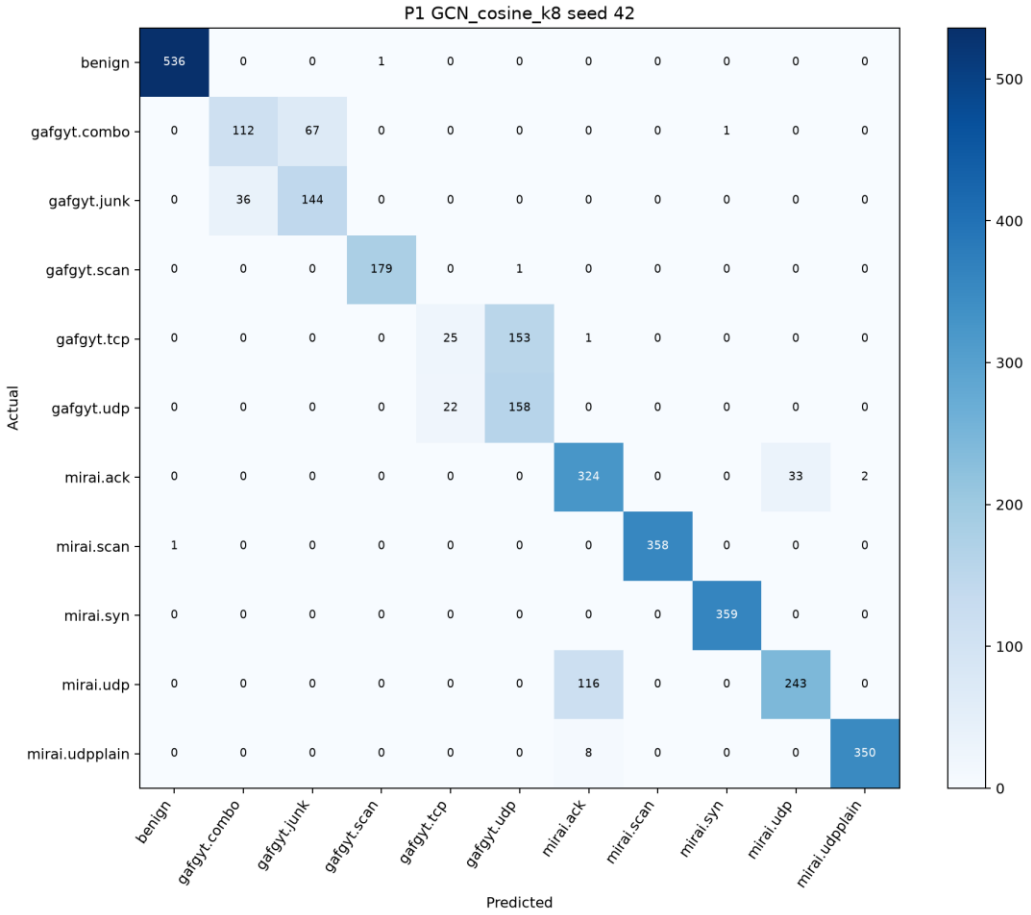


Figure 5: Fig. 5: Confusion Matrix for the Reference GCN model using Protocol 1 with Seed 42. The numbers represent results from one experiment, while Table 4 shows the results after five runs.

The largest off-diagonal counts in Figure 5 are 153 gafgyt.tcp records predicted as gafgyt.udp, 116 mirai.udp records predicted as mirai.ack, and 67 gafgyt.combo records predicted as gafgyt.junk. These are specific errors and not proof that the features do not differentiate the classes. I notice that it is harder to diagnose attacks because of the confusion in the attack subtypes. The prediction of the benignity of an attack will affect the way the attack is detected. It could be caused by the overlapping of the features or neighbors.

D. Cross-Device Performance

Based on Protocol 2, Philips Baby Monitor is not included in the training dataset. Table 5 displays the accuracy on the chosen held-out data.

Table 5: Performance of the model on a held-out Philips device (mean ± SD from 5 random seeds; %). The performance of precision, recall, and F1 is shown as a weighted average.

Model	Accuracy	Precision	Recall	F1-Score
GCN (cosine k=8)	84.06 ± 19.95	88.13 ± 14.16	84.06 ± 19.95	84.99 ± 18.26
Random Forest	99.77 ± 0.04	99.88 ± 0.06	99.77 ± 0.04	99.82 ± 0.04
XGBoost	99.59 ± 0.11	99.69 ± 0.10	99.59 ± 0.11	99.64 ± 0.11
MLP	93.72 ± 1.01	94.94 ± 1.62	93.72 ± 1.01	93.87 ± 1.08
GraphSAGE (k=8)	83.30 ± 21.64	84.16 ± 22.97	83.30 ± 21.64	81.91 ± 25.07

Both Random Forest and XGBoost algorithms achieve average accuracy above 99.5% on the particular held-out device. There is variability in the GCN algorithm, particularly in the case where seed equals 46, which is included in the

result. In Protocol 2, there are no BASHLITE instances; hence, these findings cannot be generalized to all N-BaIoT devices..

E. Cross-Device Confusion Analysis

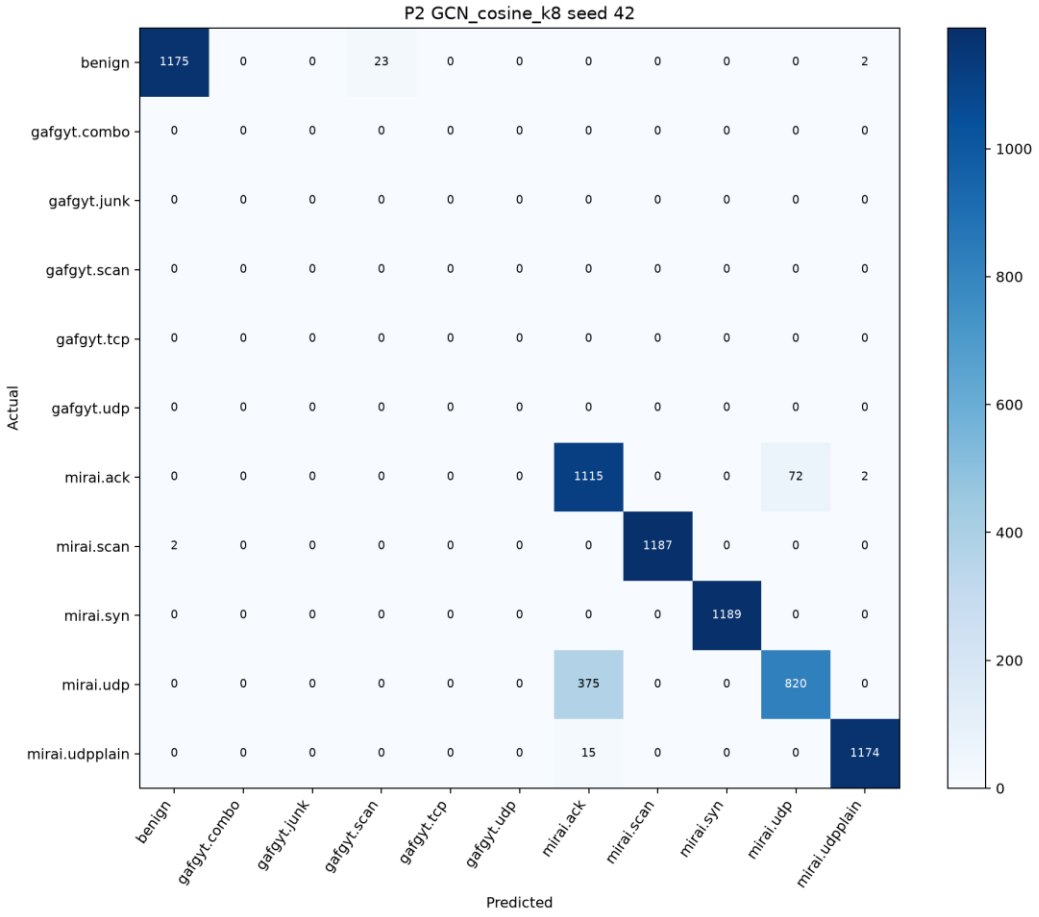


Figure 6: shows the reference GCN confusion matrix under Protocol 2 with seed 42. The zero rows, in BASHLITE indicate that certain test classes were not present. Table 5 provides the five-run performance results.

Figure 6 displays 375 mirai.udp entries that were labelled as mirai.ack, 72 mirai.ack entries that were labelled as mirai.udp and 23 benign entries that were labelled as gafgyt.scan. Mirai.udp and mirai.ack confusion happens in both of the shown protocols. This observation is a result for the model that was tested and it does not show a built-in problem, with the feature space. BASHLITE subtype confusion cannot be checked across devices because those classes do not appear in the Philips test records that were chosen.

F. Discussion

The results yielded by both the protocols deserve interpretation in addition to the raw numbers provided in the tables. The 115 statistical properties confirm excellent classifier performance on the trees under the considered settings. While GCN aggregation is likely to combine similar subtypes in appearance, this is just a reasonable explanation but not an identified causal process. Some modifications of the graph structures, shown in Table 6, show performance depending on the particular structure. Moreover, the comparison with the multilayer perceptron (MLP) confirms that the graph formation itself does not explain the difference.

The practical benefit is achieved in the form of the matched comparison and class-based diagnosis, providing an opportunity to judge if there is any benefit from graph aggregation in the considered settings. Configuration cosine k=4 and Euclidean k=8 show less observed cross-device variance than configurations cosine k=8 and k=16, as well as GraphSAGE, which contains an unstable run with seed 46. Very high standardized validation scores in the unstable run indicate sensitivity to preprocessing, even though the causality was not proved. All five runs are kept. In practice, when

using IoT, there will always be changing inventories of devices. The random splits in Protocol 1 examine how well a model generalizes to new rows from known devices, and this is a comparatively easy case compared to generalizing to a new device type. The results of tree classifiers for the selected Philips device give early evidence of transferability for known classes; nevertheless, this evidence cannot be generalized from a single hold-out device. For the determination of whether robustness applies to a wider set, a leave-one-device-out protocol has to be run on all nine N-BaIoT devices.

Different variations of flood attacks originating from the same family of botnets create voluminous but uniform traffic, varying only in the network layer protocol used (TCP or UDP) or the attack method (ACK or raw UDP). To understand if there is any relationship between feature overlap, neighborhood mixing, and optimization, further controlled experiments need to be performed. Modeling in time sequence can be a possible avenue, given that of Babenko et al.'s hybrid approach of GNN-LSTM [24].

Table 6: GCN graph sensitivity, mean ± sample SD accuracy across five seeds (%).

Graph	Protocol 1	Protocol 2
Cosine k=4	86.37 ± 0.65	92.72 ± 1.39
Cosine k=8	86.22 ± 0.39	84.06 ± 19.95
Cosine k=16	86.71 ± 2.14	82.92 ± 20.67
Euclidean k=8	86.11 ± 0.36	92.88 ± 0.82

6. Conclusion and Future Work

This paper investigated the performance of a two-layer GCN for multiclass IoT botnet detection on N-BaIoT, comparing it against Random Forest and XGBoost under a conventional in-distribution split (Protocol 1) and a cross-device split where the test device was fully excluded from training (Protocol 2). The study covered 11 traffic classes spanning benign traffic, five Mirai subtypes, and five BASHLITE subtypes across three commercial IoT devices. This study evaluates three N-BaIoT devices. Future work should extend this evaluation framework to all nine devices available in the dataset to further validate cross-device generalization.

From an engineering standpoint, tree-based classifiers remain strong baselines on the chosen N-BaIoT sample. Both of them have accuracy above 99.5% on the held-out Philips records without graph construction. This observation is valid only for the particular device being analyzed and does not imply generalization to any other device.

From a scientific standpoint, the most important class-level observation is the confusion of Mirai ACK/UDP in both protocols, including on the held-out device Philips. In Protocol 1, there is substantial confusion between BASHLITE TCP/UDP and its BASHLITE variants; yet, they are not present in the held-out Philips device in Protocol 2, and therefore are not cross-device replicated in this experiment. Overall, the findings prove that aggregation metrics hide subtype errors in the tested classifiers. Reproducibility across experiments demonstrates sensitivity to graph setup and run-time conditions.

The paired evaluation performed in this project, which consists of both random split and a fully held-out-device test, provides methodological comparison which is more deployment-oriented than the random split itself. Future IoT intrusion-detection system (IDS) evaluations should report both in-distribution and device-held-out results where possible.

Several concrete directions for further study follow from these observations. Firstly, leave-one-device-out evaluation across all nine N-BaIoT devices would be helpful to check whether cross-device robustness holds up for the devices with different traffic profiles. Replacing feature-similarity edges with communication-topology edges would help incorporate information not represented in node features and could help close the gap in GCN accuracy for flood-type subtypes. From an architectural perspective, Graph Attention Networks and Graph Transformers could learn to downweight dissimilar neighbors, thus reducing neighborhood mixing as a cause of class confusion. Temporal GNN-LSTM hybrid models and dynamic graphs are promising directions to solve flood-variant confusion through modeling of evolution rather than aggregate of one window. Finally, real-time edge deployment, as suggested by Yang et al. for Industrial IoT, would be a natural next step for architectures that achieve both good cross-device performance and efficiency.

CONFLICT OF INTEREST

There is no conflict of interest between all the authors.

DATA AVAILABILITY STATEMENT

The data analyzed in this study are from the publicly available N-BaIoT dataset described in [1].

7. References

[1] Y. Meidan, M. Bohadana, Y. Mathov, et al., “N-BaIoT: Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders,” IEEE Pervasive Computing, vol. 17, no. 3, pp. 12–22, 2018, doi: 10.1109/MPRV.2018.03367731.

[2] S. A. Bakhsh, M. A. Khan, F. Ahmed, M. S. Alshehri, H. Ali, and J. Ahmad, “Enhancing IoT network security through deep learning-powered Intrusion Detection System,” Internet of Things, vol. 24, pp. 100936, 2023, doi: 10.1016/j.iot.2023.100936.

[3] M. Hossain, R. Al Mamun Rudro, R. Razzaque, and K. Nur, “Machine Learning Approaches for Detecting IoT Botnet Attacks: A Comparative Study of N-BaIoT Dataset,” in 2024 International Conference on Decision Aid Sciences and Applications (DASA), pp. 1–7, 2024, doi: 10.1109/DASA63652.2024.10836622.

[4] A. M. N. Tarab, “Comparative Evaluation of Supervised Machine Learning Models for IoT Botnet Detection using Random Forest, XGBoost, and ANN,” Journal of Engineering, vol. 32, no. 6, pp. 22–50, 2026, doi: 10.31026/j.eng.2026.06.02.

[5] M. Rawat, A. Singh Bedi, B. Singh, S. Gupta, G. Singal, and P. Kaur, “Ensemble-Based Botnet Attack Detection and Classification Using Machine Learning Algorithms on NBaIoT Dataset,” in 2024 IEEE Region 10 Symposium (TENSYMP), pp. 1–6, 2024, doi: 10.1109/TENSYMP61132.2024.10752221.

[6] Q. Abu Al-Haija, and A. Droos, “A Comprehensive Survey on Deep Learning-Based Intrusion Detection Systems in Internet of Things (IoT),” Expert Systems, vol. 42, no. 2, pp. e13726, 2025, doi: 10.1111/exsy.13726.

[7] M. Catillo, A. Pecchia, and U. Villano, “A Deep Learning Method for Lightweight and Cross-Device IoT Botnet Detection,” Applied Sciences, vol. 13, no. 2, pp. 837, 2023, doi: 10.3390/app13020837.

[8] T. Bilot, N. El Madhoun, K. Al Agha, and A. Zouaoui, “Graph Neural Networks for Intrusion Detection: A Survey,” IEEE Access, vol. 11, pp. 49114–49139, 2023, doi: 10.1109/ACCESS.2023.3275789.

[9] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, “E-GraphSAGE: A Graph Neural Network based Intrusion Detection System for IoT,” in NOMS 2022 IEEE/IFIP Network Operations and Management Symposium, pp. 1–9, 2022, doi: 10.1109/NOMS54207.2022.9789878.

[10] L. Lin, Q. Zhong, J. Qiu, and Z. Liang, “E-GRACL: an IoT intrusion detection system based on graph neural networks,” The Journal of Supercomputing, vol. 81, pp. 42, 2025, doi: 10.1007/s11227-024-06471-5.

[11] N. Sun, L. Chen, and G. Han, “HADGA: Hierarchical Attention-Based Dynamic GNN Algorithm for IoT Botnet Detection,” IEEE Internet of Things Journal, vol. 12, no. 16, pp. 33520–33532, 2025, doi: 10.1109/JIOT.2025.3576710.

[12] A. S. Ahanger, S. M. Khan, F. Masoodi, and A. O. Salau, “Advanced intrusion detection in internet of things using graph attention networks,” Scientific Reports, vol. 15, pp. 9831, 2025, doi: 10.1038/s41598-025-94624-8.

[13] M. I. M. Tanvir, N. Y. Nadia, H. R. Rabby, M. H. Arif, K. Nur, and D. Ghose, “Spatial-temporal graph neural network with autoencoder pretraining for intrusion detection in healthcare IoT ecosystems,” Scientific Reports, vol. 16, pp. 15000, 2026, doi: 10.1038/s41598-026-45041-y.

[14] F. Taher, M. Abdel-Salam, M. Elhoseny, and I. M. El-Hasnony, “Reliable Machine Learning Model for IIoT Botnet Detection,” IEEE Access, vol. 11, pp. 49319–49336, 2023, doi: 10.1109/ACCESS.2023.3253432.

[15] Z. Wang, H. Chen, S. Yang, X. Luo, D. Li, and J. Wang, “A lightweight intrusion detection method for IoT based on deep learning and dynamic quantization,” PeerJ Computer Science, vol. 9, pp. e1569, 2023, doi: 10.7717/peerj-cs.1569.

[16] T. N. Kipf, and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks,” in International Conference on Learning Representations (ICLR), 2017.

[17] W. Villegas-Ch., J. Govea, A. Maldonado Navarro, and P. Palacios Játiva, “Intrusion Detection in IoT Networks Using Dynamic Graph Modeling and Graph-Based Neural Networks,” IEEE Access, vol. 13, pp. 65356–65375, 2025, doi: 10.1109/ACCESS.2025.3559325.

[18] O. Bondar, “Graph Neural Network-Based Intrusion Detection for IoT: Performance and Comparative Analysis,” Grail of Science, no. 54, pp. 506–514, 2025, doi: 10.36074/grail-of-science.18.07.2025.058.

[19] T. Altaf, X. Wang, W. Ni, G. Yu, R. P. Liu, and R. Braun, “GNN-Based Network Traffic Analysis for the Detection of Sequential Attacks in IoT,” Electronics, vol. 13, no. 12, pp. 2274, 2024, doi: 10.3390/electronics13122274.

[20] L. Yin, W. Chen, X. Luo, and H. Yang, “Efficient Large-Scale IoT Botnet Detection through GraphSAINT-Based Subgraph Sampling and Graph Isomorphism Network,” Mathematics, vol. 12, no. 9, pp. 1315, 2024, doi: 10.3390/math12091315.

[21] S. Yang, W. Pan, M. Li, et al., “Industrial Internet of Things Intrusion Detection System Based on Graph Neural Network,” Symmetry, vol. 17, no. 7, pp. 997, 2025, doi: 10.3390/sym17070997.

[22] F. Al Tfaily, Z. Ghalmane, M. e. A. Brahmia, H. Hazimeh, A. Jaber, and M. Zghal, “Graph-based federated learning approach for intrusion detection in IoT networks,” Scientific Reports, vol. 15, pp. 41264, 2025, doi: 10.1038/s41598-025-25175-1.

[23] D. Concejal Muñoz and A. del-Corte Valiente, “A novel botnet attack detection for IoT networks based on communication graphs,” Cybersecurity, vol. 6, pp. 33, 2023, doi: 10.1186/s42400-023-00169-6.

[24] T. Babenko, K. Kolesnikova, Y. Bakhtiyarova, et al., “Hybrid GNN–LSTM Architecture for Probabilistic IoT Botnet Detection with Calibrated Risk Assessment,” Computers, vol. 15, no. 1, pp. 26, 2026, doi: 10.3390/computers15010026.