

Hybrid Machine Learning Model for Efficient Botnet Attack Detection in IoT Environment

Vallu Srivarshitha¹, Dr. S. Jhansi Rani²

¹*MTech Student, Department of Computer Science and Systems Engineering, Andhra University College of Engineering (A), Andhra University, Visakhapatnam, India*

²*Associate Professor, Department of Computer Science and Systems Engineering, Andhra University College of Engineering (A), Andhra University, Visakhapatnam, India*

Abstract—Reliable botnet classification requires both predictive evaluation and scrutiny of the traffic records used for training. This study compares an artificial neural network (ANN), a convolutional neural network (CNN), a long short-term memory (LSTM) network, a stacked convolutional–recurrent hybrid designated ACLR, and an attention extension. Experiments use a balanced 9,999-record N-BaIoT sample and an 82,332-record UNSW-NB15 benchmark. An audit of the supplied N-BaIoT sample identifies five capture-time-related columns, repeated feature vectors and two classes with very low diversity. A cleaned profile retains 8,157 records, 110 features and nine classes. ACLR achieves 98.65% accuracy and 98.65% macro-F1 on its 1,632-record test split, leading the five neural models, although a decision tree reaches 100.00% accuracy. White-box fast gradient sign method experiments reduce ACLR accuracy to 85.72% under increase-only constraints and 64.28% under unconstrained perturbations at a standardised budget of 0.05. A Flask interface serves saved models for record and CSV classification. The findings support sample-specific classification and feature sensitivity conclusions, while motivating evaluation on separate devices and capture sessions.

Index Terms—Adversarial evaluation, botnet detection, convolutional neural network, dataset audit, Internet of Things, long short-term memory, N-BaIoT.

I. INTRODUCTION

Internet of Things devices such as cameras and sensors can be recruited into botnets that generate malicious network traffic. Mirai and Gafgyt are important examples of this threat [1], [2]. Network-based classification offers a way to flag suspicious traffic without requiring a detector on every endpoint. However, the reliability of a detector depends on

whether the evaluation measures attack behaviour or characteristics of the recording process.

Hybrid neural models combine different layer types to classify network records. Their apparent advantages should be assessed against individual neural architectures and simpler classifiers under the same protocol. Ali et al. [3] studied a hybrid ACLR approach and identified hostile attacks among the unresolved concerns. The present implementation is inspired by that study, but does not reproduce its exact architecture or validation protocol.

The study asks three questions. How do five implemented neural architectures compare on the supplied records? What changes when suspicious columns, repeated rows and low-diversity classes are excluded from the IoT sample? How sensitive are the saved neural models to gradient-based feature perturbations? The contributions are a comparison across two datasets, a reproducible sample audit, a white-box FGSM evaluation, and an application that connects saved models to user input. The cleaned task has nine classes rather than eleven, so its improved scores cannot be attributed to preprocessing alone.

II. RELATED WORK

Meidan et al. [4] introduced N-BaIoT using traffic from commercial IoT devices infected with Mirai and Gafgyt, and investigated detection with deep autoencoders. This provides the IoT attack context for the present sample. Moustafa and Slay [5] introduced UNSW-NB15 as a broader network intrusion dataset. UNSW-NB15 is used here as an architecture benchmark; results on it are not evidence of IoT-specific botnet detection.

Kolias et al. [1] and Antonakakis et al. [2] describe Mirai and the conditions enabling large botnets. McDermott et al. [6] investigated deep learning for botnet detection in IoT. Dense, convolutional and recurrent methods provide different representations [7], [8], but the usefulness of a particular representation depends on how records are constructed.

Ali et al. [3] report a hybrid detector on UNSW-NB15. This work extends the experimental perspective with an audit of a separate N-BaIoT sample and perturbation measurements. Goodfellow et al. [9] provide the FGSM formulation, while Madry et al. [10] develop the projected-gradient approach. The present reported attack results are FGSM runs; the availability of PGD in the code is not treated as evidence of a completed PGD experiment.

III. DATASETS AND EXPERIMENTAL PROTOCOL

A. Dataset profiles

The original N-BaIoT sample contains 9,999 records, with 909 records per class across benign traffic and ten Mirai/Gafgyt labels. Each record contains 115 numeric features. A stratified 80/20 row split yields 7,999 training records and 2,000 test records. This is a sample-level evaluation, not an audit of the full source archive.

The cleaned profile excludes HH_jit_L5_mean, HH_jit_L3_mean, HH_jit_L1_mean, HH_jit_L0.1_mean and HH_jit_L0.01_mean. It removes gafgyt-tcp and gafgyt-udp and then deduplicates the remaining records. Excluding the flagged columns before deduplication is necessary because these columns can distinguish otherwise repeated rows. The final profile contains 8,157 records, 110 features and nine classes, partitioned into 6,525 training and 1,632 test records. An audit intermediate count of 8,161 includes four distinct records belonging to the excluded classes and is not the training dataset size.

TABLE I DATASET PROFILES

Profile	Records	Features	Classes
N-BaIoT original	9,999	115	11
N-BaIoT cleaned	8,157	110	9
UNSW full	82,332	42	10
UNSW manual	82,332	10	10

The UNSW-NB15 experiment uses 82,332 records and 42 predictors, with ten labels and 16,467 test records. A separate ten-feature profile supports the website form. The two UNSW profiles train separate models; scores from the full profile cannot be assigned to the manual form.

B. Preprocessing and training

Target labels and bookkeeping columns are separated from the predictors. UNSW categorical fields are integer encoded, and unknown inference categories map to -1. In the current implementation, categorical mappings are fitted before the row split; fitting these mappings on training rows alone would give a stricter evaluation. StandardScaler is fitted on training rows and reused for validation, testing and inference. Feature order, label mapping and scaler are saved separately for each profile.

Training uses Adam [11] with learning rate 0.001, categorical cross-entropy, batch size 128 and a maximum of 30 epochs. Keras reserves 15% of the training partition for validation. Early stopping monitors validation loss with patience five and restores the best weights.

The row split uses seed 42. Model initialisation and all stochastic operations are not fully seeded, so the saved runs do not establish repeated-run variability. TensorFlow/Keras and scikit-learn support the implementation [12], [13].

C. Evaluation measures

Accuracy measures the proportion of correctly classified records.

Macro precision, recall and F1 assign equal weight to each class, including rare classes. For C classes, macro-F1 is computed as the mean of per-class F1 values:

$$F1_{macro} = (1/C) \sum_{c=1}^C F1_c \quad (1)$$

Scores are reported as percentages. Saved predictions and labels were checked against the saved accuracy and macro-F1 for all twenty neural model/profile combinations. These checks validate consistency of the supplied artifacts; training was not repeated during verification. A random row split does not establish generalisation to new devices, capture sessions or unseen attack categories.

IV. NEURAL MODEL ARCHITECTURES

A. Individual architectures

The artificial neural network (ANN) receives the flat feature vector and applies dense layers of 64 and 32 units with ReLU activation, followed by a class softmax. The convolutional neural network (CNN) reshapes a record to features $\times 1 \times 1$, applies two 1×1 convolutional layer with 64 and 32 channels, and then flattens the output before softmax. Its 1×1 pooling layer do not reduce dimensions. Consequently, the convolutions transform channels at individual feature positions rather than extracting neighbourhoods with a wider kernel.

The long short-term memory (LSTM) baseline receives a features $\times 1$ tensor, followed by a 64-unit LSTM, dropout of 0.5, a 32-unit dense layer and softmax. Its sequence axis represents feature positions. It should therefore not be interpreted as learning chronological packet sequences.

B. Stacked ACLR and attention extension

The stacked convolutional-recurrent hybrid, designated ACLR, chains the two convolutional stages, flattening, RepeatVector(3), dropout of 0.5, a 32-unit LSTM with ReLU activation, a 64-unit dense layer and softmax. Here, stacked means a single network trained end to end.

There is no ensemble vote or separate meta-learner. RepeatVector creates three copies of the same encoded record; these positions are not successive observations of network traffic.

The attention extension inserts an additive scoring layer between RepeatVector and the LSTM. For repeated vector h_t , its implementation computes:

$$e_t = \tanh(h_t W + b_t)$$

$$\alpha_t = \text{softmax}(e_t) \quad (2)$$

$$h'_t = \alpha_t h_t \quad (3)$$

The extension uses the default LSTM activation and omits the hybrid dropout stage. Its comparison with ACLR therefore changes more than attention alone. For the cleaned profile, ACLR has 459,689 parameters and the attention model has 463,212.

The ANN, CNN and LSTM have 9,481, 33,897 and 19,273 parameters, respectively. The common epoch ceiling does not equalise model capacity or computational cost.

V. DATA AUDIT AND ADVERSARIAL EVALUATION

A. Capture related columns and repeated vectors

Five HH_jit mean columns in the supplied sample contain values in a Unix-epoch range. Their association with recording windows motivates treating them as potential capture-related shortcuts. In the saved audit, a random forest using only these five columns achieves 91.45% accuracy and 91.45% macro-F1 across eleven labels. Using all 115 columns gives 99.50% accuracy, while excluding the flagged columns gives 90.80% accuracy and 87.75% macro-F1. These measurements demonstrate strong label information in the flagged columns within this sample; they do not establish a defect in every N-BaIoT release.

TABLE II AUDIT OF THE ORIGINAL SAMPLE

RF feature set	Count	Acc	Macro F1
All columns	115	99.50	99.50
Flagged columns only	5	91.45	91.45
Flagged columns excluded	110	90.80	87.75

After excluding the flagged columns, the 909 sampled gafgyt-udp rows contain only one distinct vector, while the 909 gafgyt-tcp rows contain three. These classes are omitted from the cleaned experiment to avoid drawing generalisation claims from their low diversity. The original audit also reports 31 duplicate rows in its unmodified representation. Exact overlap across the split is measured in a transformed feature representation and is therefore a separate quantity from that raw duplicate count.

The audit reports 18.3% exact train/test overlap in the original profile and 0.0% in the cleaned profile. The proportion of test records within distance 0.1 of a training record fall from 44.1% to 29.6%. Removing exact duplicates thus leaves substantial proximity between partitions. The most informative next evaluation would retain device/session identifiers and separate those groups across training and testing.

B. Feature space threat model

The fast gradient sign method (FGSM) uses white-box access to the model and true labels to increase

classification loss. In standardised coordinates, the unconstrained update is [9]:

$$x_{adv} = x + \varepsilon \text{sign}(\nabla_x L(\theta, x, y)) \quad (4)$$

The saved experiments sweep ε over 0, 0.01, 0.02, 0.05, 0.10, 0.20 and 0.50. These are budgets in scaled feature units rather than bytes or packets. The unconstrained attacker may increase or decrease any feature. The increase-only variant clips generated values using training-set extrema and prevents decreases in every feature, not only traffic volume variables.

These constraints do not preserve relationships among traffic statistics and do not establish that equivalent packet traces can be generated. For out-of-range test coordinates, clipping may also change a coordinate beyond the nominal budget.

We therefore describe this experiment as increase-only feature perturbation, rather than physically validated traffic evasion. The zero-budget constrained reference passes through the same constraint procedure.

The saved robustness score measures relative accuracy retention across the six nonzero budgets. With a_ε denoting accuracy under the selected constraint procedure, and $E = \{0.01, 0.02, 0.05, 0.10, 0.20, 0.50\}$, it is:

$$R = 100 \times [(1/6) \sum_{\varepsilon \in E} a_\varepsilon] / a_0 \quad (5)$$

A larger score indicates greater retention over this particular sweep. It is not an average over all seven points or a guarantee against other attackers. Projected gradient descent (PGD) is implemented, but no PGD result is used in this paper. No adversarial retraining or defence is claimed.

VI. RESULTS AND DISCUSSION

A. Neural classification results

Table III compares original and cleaned N-BaIoT results. ACLR reaches 84.80% accuracy and 81.65% macro-F1 on the original eleven-class task. On the cleaned nine-class task it reaches 98.65% for both measures, leading the five neural architectures. The attention extension reaches 97.86% accuracy and 97.85% macro-F1. Fig. 1 visualises the protocol-dependent accuracy.

TABLE III N BAIOT NEURAL PERFORMANCE

Model	Orig acc	Orig F1	Clean acc	Clean F1
ANN	82.45	79.32	89.83	89.62
CNN	83.15	79.96	92.22	92.19
LSTM	69.15	62.8	70.77	69.82
ACLR	84.8	81.65	98.65	98.65
Attention	83.85	80.75	97.86	97.85

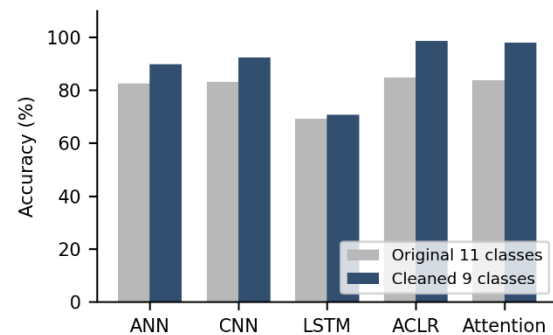


Fig. 1. Accuracy on the original and cleaned N-BaIoT tasks. The class sets and record populations differ.

The increase from 84.80% to 98.65% is 13.85 percentage points, but combines feature exclusion, deduplication and removal of two difficult low-diversity classes. It does not quantify the independent causal effect of leakage removal. The attention result also does not show that attention improves ACLR, since its clean accuracy is lower and its builder changes activation and dropout.

On the original N-BaIoT task, collapsing every non-benign category to attack gives ACLR 99.80% binary accuracy. Only 181 of 2,000 test records are benign, so always predicting attack would already reach 90.95%. Binary accuracy and multiclass accuracy answer different questions and must be reported separately.

TABLE IV UNSW NB15 FULL PROFILE

Model	Acc	Macro P	Macro R	Macro F1
ANN	85.36	55.53	47.5	48.04
CNN	85.17	57.84	49.29	49.97
LSTM	83.41	41.29	40.84	40.58
ACLR	86.66	67.21	52.53	54.63
Attention	86.93	60.92	51.26	52.29

On UNSW-NB15, attention gives the largest accuracy, 86.93%, while ACLR gives the largest macro-F1, 54.63% (Table IV). The divergence reflects class imbalance: the frequent Normal and Generic classes dominate accuracy, while rare labels carry equal weight in macro-F1. The test partition contains only nine Worms records, and several rare classes have no correct predictions. High aggregate accuracy therefore does not imply comparable detection of every attack category.

B. Classical baseline comparison

TABLE V CLASSICAL BASELINE ACCURACY

Classifier	Original	Cleaned
Random forest 200 trees	99.55	99.88
Decision tree	99.50	100.00
k nearest neighbours k 5	95.60	95.59
Stacked ACLR	84.80	98.65

The decision tree and random forest exceed ACLR accuracy on the cleaned random split. Table V therefore does not support overall superiority of the hybrid.

The combination of perfect decision-tree accuracy and remaining close neighbours motivates a harder grouped split.

Classical models were not included in the saved attack experiment, so a robustness advantage over these classifiers is also unestablished.

C. FGSM sensitivity

TABLE VI CLEANED PROFILE FGSM RESULTS

Model	Clean acc	Inc acc	Free acc	R inc	R free
ANN	89.83	83.76	71.26	72.10	54.73
CNN	92.22	74.88	50.43	61.63	47.09
LSTM	70.77	38.54	23.28	48.77	34.76
ACLR	98.65	85.72	64.28	73.04	54.88
Attention	97.86	77.39	59.74	66.27	51.67

In Table VI, increase and free accuracy use $\epsilon = 0.05$; R uses the full nonzero sweep. ACLR retains 85.72% accuracy under increase-only perturbations and 64.28% under unconstrained perturbations. These are the highest absolute accuracies among the five neural models at that budget. Relative retention differs much less: ACLR exceeds ANN by approximately 0.94 points for the increase-only score and 0.15 points for the unconstrained score.

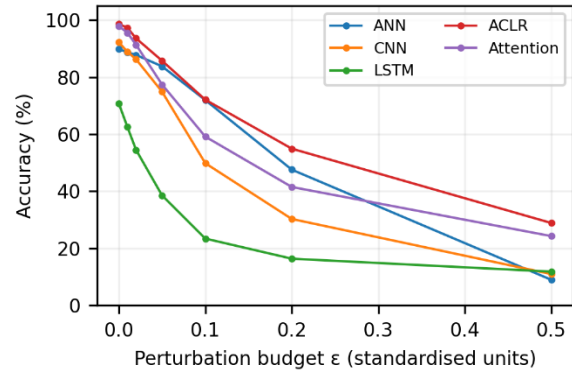


Fig. 2. Accuracy under increase-only FGSM feature perturbations on the cleaned profile.

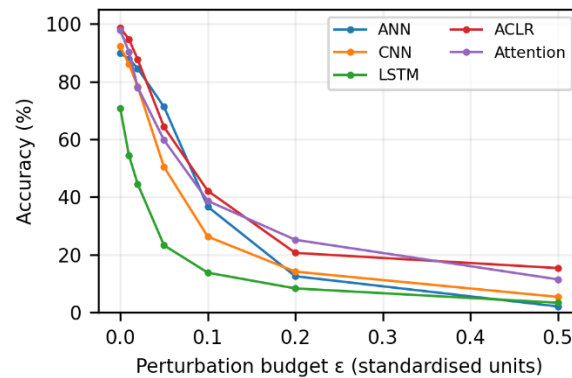


Fig. 3. Accuracy under unconstrained FGSM feature perturbations on the cleaned profile.

Figs. 2 and 3 show declining accuracy as the budget grows. At $\epsilon = 0.50$, increase-only ACLR accuracy is 28.86%, substantially below its clean score. The saved experiment thus establishes sensitivity even for the strongest neural classifier in this comparison. Small differences in relative retention require repeated runs before a stable ranking is asserted. The results do not isolate an architectural reason for robustness or verify practical packet-level evasion.

VII. WEB APPLICATION AND VERIFICATION

The Flask application separates training from inference. Saved weights and preprocessing artifacts are loaded at first use and cached. The frontend uses HTML, CSS, JavaScript, Bootstrap and Jinja2 templates. Account registration and login precede protected prediction pages. The manual form accepts ten UNSW features, while CSV classification supports the full UNSW, original N-BaIoT and cleaned N-

BaIoT schemas. The algorithm page displays saved model/profile results.

The form and batch paths default to the attention model. Therefore, the 98.65% cleaned ACLR result is not the accuracy of the manual form. The selected-profile attention model has 77.57% saved accuracy and 33.64% macro-F1. Three of six built-in examples were classified correctly in a demonstration check; this small check is not an overall accuracy estimate.

All sixteen documented functional cases and seven-page checks passed against a temporary database. All twenty saved neural model/profile combinations loaded and produced sample inference. Live HTTP checks confirmed registration, login and the main application pages. Visual browser rendering and mobile layout were not verified. Functional success establishes that the application processes input and returns results; it does not establish that each classification is correct.

The present interface accepts precomputed feature records. It does not capture live network traffic or compute the required features from packets. CSV parsing currently converts invalid numeric values to zero, and stronger finite-value/range validation is needed. Model and preprocessing version checks, cross-site request forgery (CSRF) protection and login rate limiting are appropriate additions before public deployment. No inference latency or operational deployment claim is made.

VIII. CONCLUSION AND FUTURE SCOPE

This study compared five neural architectures, audited a supplied N-BaIoT sample and measured white-box FGSM sensitivity. ACLR achieved 98.65% accuracy and macro-F1 on the cleaned nine-class random split, leading the neural models while remaining below decision-tree and random-forest accuracy. Its performance fell to 85.72% and 64.28% under the two feature perturbation settings at $\epsilon = 0.05$. The audit and attack measurements show why classification scores should be interpreted with their data preparation and threat model. The Flask application provides a working interface to the saved classifiers.

Future work will retain device and capture-session metadata for grouped splits, extend sampling, and evaluate multiple training seeds. Chronological traffic sequences, matched-capacity models and controlled attention ablations can clarify the architectural

comparison. Traffic-aware constraints, additional PGD experiments, adversarial training and attacks on classical classifiers can strengthen robustness evaluation. Live feature extraction, stricter input validation, calibrated confidence thresholds and downloadable prediction reports can extend the application.

REFERENCES

- [1] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *IEEE Comput.*, vol. 50, no. 7, pp. 80–84, 2017.
- [2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai botnet," in *Proc. 26th USENIX Security Symp.*, pp. 1093–1110, 2017.
- [3] M. Ali, M. Shahroz, M. F. Mushtaq, S. Alfarhood, M. Safran, and I. Ashraf, "Hybrid machine learning model for efficient botnet attack detection in IoT environment," *IEEE Access*, vol. 12, pp. 40682–40699, 2024, doi: 10.1109/ACCESS.2024.3376400.
- [4] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, D. Breitenbacher, A. Shabtai, and Y. Elovici, "N-BaIoT: Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Comput.*, vol. 17, no. 3, pp. 12–22, 2018.
- [5] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems," in *Proc. Military Communications and Information Systems Conf. (MilCIS)*, Canberra, Australia, pp. 1–6, 2015.
- [6] C. D. McDermott, F. Majdani, and A. V. Petrovski, "Botnet detection in the Internet of Things using deep learning approaches," in *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, Rio de Janeiro, Brazil, pp. 1–8, 2018.
- [7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

- [9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2015.
- [10] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2018.
- [11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learning Representations (ICLR)*, San Diego, CA, USA, 2015.
- [12] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Systems Design and Implementation (OSDI)*, pp. 265–283, 2016.
- [13] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.