RESEARCH ARTICLE OPEN ACCESS

A Hybrid Deep Learning Framework for Real-Time Botnet Attack Detection in IoT Networks

Vadde Sai Sharan, Vijay Krishna M, Yashaswini BS
Department of Computer Science and Engineering (AI & ML)
Don Bosco Institute of Technology
Bengaluru, India

***Abstract*—** Internet of Things deployment has outrun the security practice applied to it, leaving a large population of under-monitored, resource-constrained endpoints available to botnet families such as Mirai and BASHLITE. Detection models built around a single network architecture tend to capture either the spatial structure of a traffic flow or its temporal structure, but rarely both at once. This paper describes BotSentry, a framework in which convolutional, recurrent, LSTM and dense layers are chained into one trainable network that consumes flow-level features and assigns each flow to one of three operational tiers: Normal, Suspicious or Botnet. The model is not confined to offline scoring. It is embedded in a Flask/Socket.IO application backed by MySQL that reads the active network interface, derives flow features from live traffic counters, and pushes statistics, per-device status and threat alerts to a browser dashboard without a page reload, while retaining a conventional upload-and-analyse path for pre-captured traffic. Training and inference share one preprocessing pipeline - cleaning, encoding, scaling and SMOTE-based balancing of the training partition - so the transformation applied to a live flow is by construction the transformation the model was fitted under. Section VI reports accuracy, precision, recall, F1-score, ROC-AUC and PR-AUC on a held-out test set. The contribution lies less in the individual components than in their arrangement: a hybrid spatial-sequential classifier delivered as a working capture-to-alert system rather than a notebook result.

***Keywords*—** *IoT Security; Botnet Detection; Hybrid Deep Learning; ANN; CNN; RNN; LSTM; Network Traffic Analysis; Cybersecurity; Intrusion Detection System (IDS); DDoS Detection.*

I. INTRODUCTION

The security practice applied to consumer IoT hardware has not kept pace with its deployment. Smart cameras, home routers, thermostats and sensors ship with default credentials, receive firmware updates irregularly if at all, and carry too little compute headroom to run a meaningful on-device defence. The result is a large and reachable population of machines that can be conscripted. Mirai and BASHLITE demonstrated what follows: fleets of compromised consumer hardware directed at distributed denial-of-service targets, used to harvest credentials, and used as a foothold for lateral movement [1].

Catching that behaviour means watching the shape of the traffic rather than its contents. Packet and byte counts, flow duration, the mix of protocols in use, how often a device opens a connection - these remain visible when payloads are encrypted or proprietary, and they carry enough signal for a classifier to separate ordinary device chatter from command-and-control and attack traffic. The autoencoder work on N-BaIoT [1] and the flow-labelled Bot-IoT corpus [2] both rest on this premise.

What most of the published work then does, however, is stop at the split. A static capture is divided into training and test partitions, a score is reported, and the model is never attached to anything that captures traffic, stores a result, or raises an alert. The distance between that and a tool a defender could actually run is considerable, and it is rarely acknowledged.

BotSentry is our attempt to close it. The system locates and reads the active local interface without being told which one to use; it derives flow-level features from counters and connection state observed on that interface; it classifies each flow with a hybrid CNN-RNN-LSTM-ANN network trained under precisely the preprocessing applied at inference; and it presents the outcome through a dashboard updated over WebSockets, with a device inventory, an alert feed and a detection history held in MySQL.

A. Objectives

- To design a system that is able to identify botnet attacks in IoT network traffic with utilizing machine learning methods.
- To apply deep learning algorithms (ANN, CNN, RNN, LSTM) in order to process traffic data effectively.
- To identify malicious traffic patterns through the extraction of features from IoT network datasets.
- To develop a user-friendly interface that enable users to upload network traffic data sets and detect attack results and simultaneously provides a live monitoring view for tracking attacks in real-time.
- s• To improve IoT network security by accurately detecting botnet attacks and reducing cyber threats, with fewer false positives than single-model baselines.

B. Contributions

- A single trainable network in which a CNN stage extracts local structure across the feature sequence, an RNN stage models short-range dependence, an LSTM stage models longer-range dependence, and dense layers perform the final classification.
- A preprocessing contract - feature ordering, encoding and scaling - persisted at training time and reloaded unchanged on both the live and the upload prediction paths, which removes the possibility of train/serve skew.
- A graded three-tier output, Normal / Suspicious / Botnet, governed by two confidence thresholds an operator can adjust without retraining.
- A working prototype offering both continuous automatic monitoring and a manual upload-and-analyse workflow.
- An end-to-end evaluation on a labelled intrusion dataset, reported in Section VI with standard metrics and placed alongside closely related hybrid models.

II. RELATED WORK

Table I sets out representative recent work on hybrid deep learning for IoT botnet and intrusion detection. The N-BaIoT work of Meidan et al. [1] trained one deep autoencoder per device on statistical snapshots of its traffic, flagging Mirai and BASHLITE activity as reconstruction error, and in doing so produced one of the field's most reused datasets. Koroniotis et al. [2] contributed Bot-IoT, whose normal and attack flows were labelled in a purpose-built testbed and which has since become a standard reference point for flow-based detection.

Attention has since shifted toward architectures that model space and time together. SkipGateNet [3] is a CNN-LSTM with learnable skip connections, kept light enough for constrained IoT and fog nodes. Arun et al. [4] paired a CNN-LSTM classifier with a blockchain ledger so that reported performance metrics could be independently verified, testing on IoT-23 with SMOTE balancing. Nazir et al. [5] evaluated a hybrid CNN-LSTM across IoT-23, N-BaIoT and CICIDS2017 to span a wider range of attack types. An attention-augmented CNN-BiLSTM on N-BaIoT has been reported at roughly 99% accuracy [6], and an LSTM-CNN on Bot-IoT above 99% with few false positives under normal traffic, although that margin narrows once adversarial perturbation is introduced [7].

The nearest neighbour to the present work is ACLR [8], a stacking ensemble whose four base learners - ANN, CNN, LSTM and RNN - are trained separately and whose outputs are combined by a meta-learner. Evaluated on UNSW-NB15 [9], the base learners were reported at test accuracies of 0.7568, 0.9440, 0.9651 and 0.9522 respectively, and the stacked model at 0.9698, rising to 0.9749 under five-fold cross-validation, with a ROC-AUC of 0.9934 and a PR-AUC of 0.9950. The same study placed ACLR ahead of its own paired combinations and of several earlier detectors on that dataset, while noting that the ensemble costs more to train, in wall-clock time and in arithmetic operations, than any of its parts. We adopt the same four component types but arrange them differently: instead of four independent models feeding a meta-learner, BotSentry chains them in sequence - CNN to RNN to LSTM to dense - as one network trained end to end. We also carry the evaluation out of the notebook. Where [8] reports results from a Colab environment, BotSentry is exercised as a running service with live monitoring and an upload path.

Three gaps recur across this literature, including in its strongest results. Evaluation stays offline, with no path from capture to alert that a user could operate. Output is binary or follows the dataset's own class labels rather than the graded severity a non-specialist needs in order to triage. And the equivalence between training-time and inference-time preprocessing is assumed rather than demonstrated, which is where silent mismatch enters deployed systems. BotSentry is built around these three points, using [8] as its principal comparison.

TABLE I

Comparison of Related Hybrid Botnet/Intrusion Detection Approaches

Approach	Technique	Strength	Limitation
Meidan et al. [1] (N-BaIoT)	Per-device deep autoencoder	Established a widely-used IoT botnet dataset/benchmark	Offline evaluation; per-device model, not a live pipeline
Koroniotis et al. [2] (Bot-IoT)	Realistic testbed, flow labeling	High-quality labeled flow dataset for forensic analytics	Dataset contribution only, not a deployed detector
Alshehri et al. [3] (SkipGateNet)	Lightweight CNN-LSTM with skip connections	Resource-efficient for constrained IoT/Fog nodes	Evaluated offline on benchmark data
Arun et al. [4]	CNN-LSTM + blockchain-logged metrics	Verifiable, tamper-evident performance logging	No real-time capture/alert pipeline described
Nazir et al. [5]	Hybrid CNN-LSTM, multi-dataset	Broad attack-type coverage (IoT-23, N-BaIoT,	Offline, dataset-only evaluation

Approach	Technique	Strength	Limitation
		CICIDS2017)	
Attention CNN-BiLSTM [6]	Attention-augmented CNN-BiLSTM	High reported accuracy on N-BaIoT	No live deployment; single-dataset benchmark
LSTM-CNN [7]	LSTM + CNN on Bot-IoT	Very high accuracy under normal conditions	Accuracy drops under adversarial traffic
Ali et al. [8] (ACLR)	Stacked ANN+CNN+LSTM+RNN ensemble	0.9698 test accuracy, 0.9934 ROC-AUC on UNSW-NB15; outperforms every individual sub-model	Notebook-only (Google Colab) evaluation; high training time/computational cost; no live pipeline
BotSentry (this work)	Sequential CNN→RNN→LSTM→ANN hybrid, live + upload pipeline	Live capture, 3-tier risk output, shared train/serve preprocessing, MySQL history + alerting, user upload workflow	Live flow features derived from OS counters rather than deep packet inspection

III. PROPOSED METHODOLOGY

A. System Overview

The pathway through the system goes from local Wi-Fi devices to the router then onto the active interface, through feature extraction and preprocessing, into the hybrid classifier, and ends up as a risk score to the dashboard, the MySQL history and the alert feed. Interface selection is automatic: the system scans non-loopback interfaces, keeps those that are up and hold an IPv4 address, and prefers a Wi-Fi adapter when it is a present option. If nothing usable is found it reports that fact rather than displaying invented figures.

B. Feature Extraction

Ten features are derived per active connection: source and destination port, protocol as a TCP/UDP one-hot pair, packets and bytes attributable to the observation window, packets per second, bytes per second, flow duration, and a flag for whether the connection is ESTABLISHED. Packet and byte deltas come from the operating system's interface I/O counters, sampled at the beginning and end of each monitoring cycle, so no feature is synthetic.

C. Preprocessing

Numeric columns are imputed with the median and categorical columns with the mode; duplicate rows are dropped; categorical fields are integer-encoded. A StandardScaler is fitted on the training split and written to disk, and that same fitted object is loaded at inference, so a live flow is scaled by the statistics the model was trained under. SMOTE is applied after the split and to the training partition alone, which addresses the imbalance typical of raw captures - where attack traffic is the minority class - without leaking synthetic samples into the test set.

D. Hybrid Model Architecture

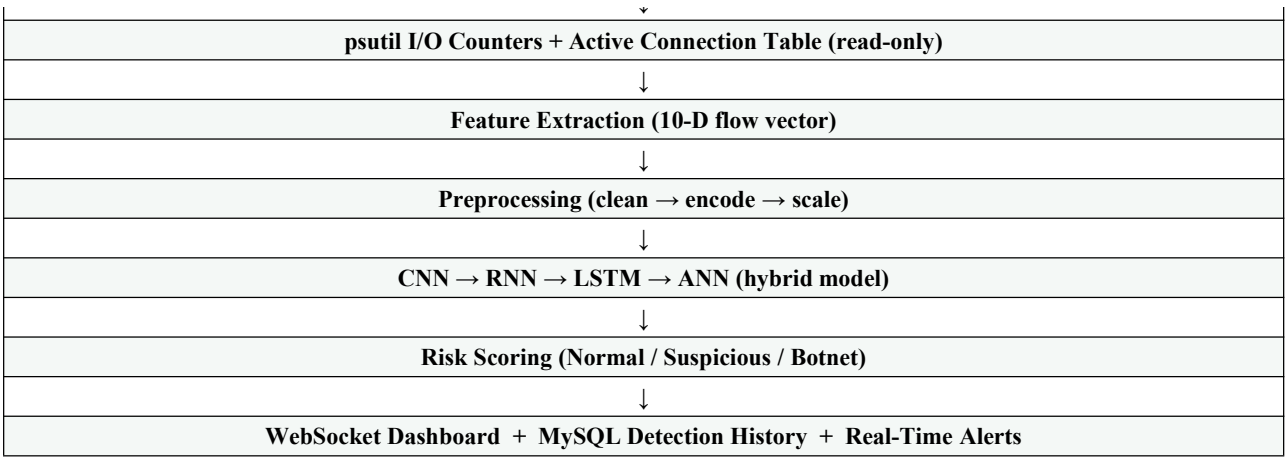
The ten features are reshaped into a single-channel sequence and passed through two 1-D convolutional blocks (32 and 64 filters, kernel size 3, ReLU, batch normalisation, max-pooling), which pick up local structure across the feature sequence in the role CNN stages play in [1], [3]-[8]. A SimpleRNN layer of 64 units with return_sequences enabled follows, capturing short-range dependence, then an LSTM of 64 units for longer range, matching the temporal role LSTM takes in the surveyed hybrids including the LSTM learner within ACLR [8]. The distinction from [8] is structural. There, four components are independent models whose predictions a meta-learner combines; here the LSTM output passes into dense layers of 64 and 32 units with ReLU and dropout, then a sigmoid unit producing a botnet probability, with the whole chain trained as a single network. The trade is the accuracy premium stacking usually buys, given up for a lighter model with shorter training time and smaller inference cost - the right side of that trade for a system that must also sustain a continuous monitoring loop.

E. Risk Scoring

Two thresholds, τ_susp and τ_bot (defaulting to 0.45 and 0.75), turn the sigmoid output into three tiers. At or above τ_bot the flow is labelled BOTNET at high risk, between the two it is SUSPICIOUS at medium risk, and below τ_susp it is NORMAL. An operator working a busy feed can then triage by severity rather than treat one binary flag as the whole answer.

System Architecture Diagram

Local Wi-Fi / IoT Devices
↓
Router → Auto-detected Network Interface



IV. SYSTEM ARCHITECTURE / IMPLEMENTATION

The backend is Python, Flask + Flask-SocketIO, Flask-Login and Flask-SQLAlchemy. A daemon thread runs the monitoring loop: it reads the interface I/O counters and the active connection table every cycle, builds feature vectors from the delta against the previous cycle, call the model, update device and detection state in memory and in MySQL, and emits the WebSocket events (live_stats, live_flows, device_update, botnet_alert) that the frontend consumes without reloading. Monitoring starts when the application boots (there is no interface-selection screen) and Start, Pause and Stop controls are always available.

MySQL holds four tables. users stores credentials under Werkzeug's password hashing. detection_history records every Suspicious or Botnet event with source and destination IP, protocol, packet and byte counts, prediction, confidence, risk level and timestamp. devices keeps per-IP status, risk level and traffic totals. system_logs captures authentication and operational events.

The frontend is server rendered across 3 views - Dashboard, Devices and History - on a light palette, with status chips, continually updated traffic table and threat-alert feed, all driven by a Socket.IO client. Training is run from a standalone script which persists the model, along with its scaler, its feature-column ordering and its label encoder, so the prediction path loads those artefacts rather than reimplementing the transformations and cannot drift from them.

Fig. 1. BotSentry end-to-end pipeline, from local traffic capture to dashboard, history, and alerting.

V. EXPERIMENTAL SETUP

A. Dataset

[TO COMPLETE: name the dataset actually used for training, number of records and benign/attack sample split. If UNSW-NB15 [9] was used - 82,332 flow records, 45 features, nine categories: Normal, Generic, Exploits, Fuzzers, DoS, Reconnaissance, Analysis, Backdoor, Shellcode and Worms - Table III becomes directly comparable with [8]. If Bot-IoT, N-BaIoT or IoT-23 was used instead, state that here.]

B. Train/Test Split and Class Balancing

The data were divided using stratified sampling into training and held-out test sets, with class ratios maintained. [TO COMPLETE: state what split was used - 80/20, or 70/30; the latter is used in [8], and would make the comparison in Table III tighter.] SMOTE was applied after the split, to the training partition only.

C. Tools and Software Environment

The following tools were used for model development: TensorFlow/Keras, scikit-learn, imbalanced-learn libraries for preprocessing and class balancing, and Pandas/NumPy for data manipulation. Offline flow generation used Wireshark and CICFlowMeter tools, and the live path is monitored using psutil interface counters.. Experimentation was carried out in Jupyter and Google Colab, and the Flask application in VS Code. [TO COMPLETE: library versions, CPU/GPU and RAM for the reported run.]

D. Evaluation Metrics

Results are reported as accuracy, precision, recall, F1-score, ROC-AUC and PR-AUC on the held-out test set, together with a confusion matrix and a full classification report. Accuracy is deliberately not reported alone: on a class-imbalanced capture, a model that predicts the majority class everywhere still scores well on it.

E. Live Deployment Test

Beyond the offline evaluation, the trained model was attached to the live pipeline and observed for [TO COMPLETE: duration] on a local network carrying ordinary household and laboratory traffic, to confirm that prediction, alerting and history logging behave correctly on

real, unlabelled input. No botnet traffic was introduced into this test.

VI. RESULTS AND DISCUSSION

Table II gives the offline test-set results for the trained hybrid model.

TABLE II

Offline Evaluation Results on the Held-Out Test Set

Metric	Value
Accuracy	[to complete]
Precision	[to complete]
Recall	[to complete]
F1-score	[to complete]
ROC-AUC	[to complete]
PR-AUC	[to complete]
False Positive Rate	[to complete]

TABLE III

Contextual Comparison with Reported Literature Results

Method	Dataset	Reported Accuracy	ROC-AUC / PR-AUC
ACLR stacking ensemble [8]	UNSW-NB15	0.9698 (test); 0.9749 (5-fold CV)	0.9934 / 0.9950
Individual ANN sub-model [8]	UNSW-NB15	0.7568	—
Individual CNN sub-model [8]	UNSW-NB15	0.9440	—
Individual LSTM sub-model [8]	UNSW-NB15	0.9651	—
Individual RNN sub-model [8]	UNSW-NB15	0.9522	—
Attention CNN-BiLSTM [6]	N-BaIoT	≈ 99%	—
LSTM-CNN [7]	Bot-IoT	99.87% (degrades under adversarial conditions)	—
BotSentry (this work)	[your dataset]	[to complete]	[to complete]

Table III places the BotSentry result beside accuracies reported for comparable hybrid architectures. Those figures come from different datasets, splits and threat models, so the table should be read as context rather than as a controlled comparison. A controlled comparison would require re-running each baseline on the identical dataset and split used here, which we note below as follow-up work.

[TO COMPLETE: two to four sentences on the confusion matrix - which classes are confused with which, whether false negatives or false positives dominate, and why. If any feature-importance analysis was run, name the features most responsible.]

[TO COMPLETE: what the live test showed - flows processed, alerts raised, any false positives and their likely cause - and whether the dashboard, history log and alert feed stayed consistent with the underlying detections.]

VII. CONCLUSION

This paper has described BotSentry, a botnet detection framework for IoT and Wi-Fi networks that joins a hybrid CNN-RNN-LSTM-ANN classifier to a live capture-to-alert pipeline, a MySQL detection history, and a dashboard requiring no manual interface configuration. Much of the surveyed literature stops at an offline score. BotSentry instead runs the trained model inside a system where training-time and inference-time preprocessing are the same persisted objects, which removes a class of deployment failure that is easy to introduce and hard to notice.

Several extensions suggest themselves: optional raw packet capture for richer flow features, simultaneous monitoring across multiple interfaces, and email or SMS alert delivery. The more substantial piece of follow-up work is a controlled comparison - retraining ACLR [8] and the baselines from [1] and [3]-[7] on the dataset and split used here, so that Table III can be replaced with a like-for-like benchmark.

REFERENCES

[1] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders," IEEE Pervasive Computing, vol. 17, no. 3, pp. 12–22, 2018.

[2] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset," Future Generation Computer Systems, vol. 100, pp. 779–796, 2019.

[3] M. S. Alshehri et al., "SkipGateNet: A Lightweight CNN-LSTM Hybrid Model with Learnable Skip Connections for Efficient Botnet Attack Detection in IoT," IEEE Access, 2024, doi: 10.1109/ACCESS.2024.3371992.

[4] A. R. Arun, A. R. de Souza, S. Sairam, V. Vani, and N. Karthik, "IoT Botnet Detection using a Hybrid of CNN-LSTM with Blockchain," in Proc. 2024 Int. Conf. Computing and Intelligent Reality Technologies (ICCIRT), IEEE, 2024, pp. 293–297.

[5] A. Nazir, J. He, N. Zhu, S. S. Qureshi, S. U. Qureshi, F. Ullah, A. Wajahat, and M. S. Pathan, "A deep learning-based novel hybrid CNN-LSTM architecture for efficient detection of threats in the IoT ecosystem," Ain Shams Engineering Journal, vol. 15, no. 7, 2024, doi: 10.1016/j.asej.2024.102777.

[6] "Efficient IoT Intrusion Detection with an Improved Attention-Based CNN-BiLSTM Architecture," arXiv:2503.19339, 2025.

[7] "A high performance hybrid LSTM CNN secure architecture for IoT environments using deep learning," Scientific Reports, 2025.

[8] 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.

[9] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in Proc. Military Communications and Information Systems Conference (MilCIS), Nov. 2015, pp. 1–6.