# Anomaly Detection in IoT Environments: A Preliminary Study

Janya N Bysani

March 17, 2025

**Abstract**

This paper examines various machine learning models for anomaly detection in IoT environments. Using the CICIoT2023 dataset as a representative example, this paper detail how data was collected and pre-processed, discusses feature selection, and compare multiple classification algorithms. The results show that a binary classification approach (benign vs. attack) achieves high accuracy, recall, precision, and F1-scores across the chosen models.

## 1 Introduction

IoT devices have become increasingly prevalent, raising security concerns due to their limited computing resources and diverse attack surfaces. In this work, we focus on evaluating different machine learning models for anomaly (attack) detection using the **CICIoT2023** [1] dataset. This dataset originates from a monitored IoT network that includes smart home devices such as Alexa-enabled assistants, cameras, speakers, power outlets, and lightbulbs, along with Raspberry Pi 4 devices configured as malicious actors.

Network traffic in this environment was captured using Wireshark. The raw `pcap` files were then processed and labeled to reflect specific attack types (e.g., DDoS floods, web-based attacks), and converted into CSV files for convenient machine learning analysis. This setup emulates real-world IoT deployments, where benign and malicious devices coexist, providing a realistic testbed for intrusion detection.

## 2 Related Works

In recent years, Internet traffic and device usage have surged, particularly with the rise of IoT (Internet of Things) applications [2]. Many IoT use cases demand low-latency, high-frequency communication, often resulting in massive numbers of small packets that are challenging to manage. Further complicating matters is the fact that a large proportion of IoT traffic involves personal or sensitive data, intensifying cybersecurity risks. Among these risks, malware stands out as a major threat, capable of posing as legitimate content to disrupt device functionality.

To combat these threats, researchers have devoted considerable effort over the past few years to developing network-based intrusion detection systems (NIDS) [3, 4, 5], which monitor network activity and filters out malicious traffic, this traffic is mirrored to NIDS devices for intrusion detection. In more recent work, [6] built an Enhanced Healthcare Monitoring System (EHMS) that gathers both patient biometric data and network flow metrics, creating a dataset of 16 thousand normal and attack records in healthcare contexts. They then tested various ML-based intrusion detection algorithms—such as Random Forests, KNN, SVM, and ANN—on this dataset. Similarly, [7] proposed an intrusion detection system using convolutional neural networks (CNNs), which classifies intrusions into four severity levels: critical, informal, major, and minor. While these methods effectively identify suspicious activity, many rely on external devices or heavyweight computation in the control plane for detection, which can introduce delays—particularly in latency-sensitive IoT scenarios.

Such extra steps—sending traffic to a remote detection system and then applying rules—add latency between network switches and the security layer. In latency-critical IoT environments, this overhead can be problematic, highlighting the need for lightweight, real-time anomaly detection and mitigation solutions at or near the network edge.

## 3 Proposed Approach

The primary goal in this work is to leverage supervised machine learning methods for accurate and efficient intrusion detection in IoT environments. This section outlines the steps from data acquisition through feature selection, model choice, and the decision to use binary classification.

## 3.1 Step 1: Data Acquisition

I use the labeled CICIoT2023 [1] CSV files generated from Wireshark captures in the IoT network. Each CSV corresponds to traffic sessions with annotations (e.g., benign or specific attack types). This structure allows for straightforward parsing and feature extraction for machine learning models.

| # | Feature | Description |
|---|---|---|
| 1 | ts | Timestamp |
| 2 | flow duration | Duration of the packet's flow |
| 3 | Header Length | Header Length |
| 4 | Protocol Type | IP, UDP, TCP, IGMP, ICMP, Unknown (Integers) |
| 5 | Duration | Time-to-Live (ttl) |
| 6 | Rate | Rate of packet transmission in a flow |
| 7 | Srate | Rate of outbound packets transmission in a flow |
| 8 | Drate, | Rate of inbound packets transmission in a flow |
| 9 | fin flag number | Fin flag value |
| 10 | syn flag number | Syn flag value |
| 11 | rst flag number | Rst flag value |
| 12 | psh flag numbe | Psh flag value |
| 13 | ack flag number | Ack flag value |
| 14 | ece flag numbe | Ece flag value |
| 15 | cwr flag number | Cwr flag value |
| 16 | ack count | Number of packets with ack flag set in the same flow |
| 17 | syn count | Number of packets with syn flag set in the same flow |
| 18 | fin count | Number of packets with fin flag set in the same flow |
| 19 | urg coun | Number of packets with urg flag set in the same flow |
| 20 | rst count | Number of packets with rst flag set in the same flow |
| 21 | HTTP | Indicates if the application layer protocol is HTTP |
| 22 | HTTPS | Indicates if the application layer protocol is HTTPS |
| 23 | DNS | Indicates if the application layer protocol is DNS |
| 24 | Telnet | Indicates if the application layer protocol is Telnet |
| 25 | SMTP | Indicates if the application layer protocol is SMTP |
| 26 | SSH | Indicates if the application layer protocol is SSH |
| 27 | IRC | Indicates if the application layer protocol is IRC |
| 28 | TCP | Indicates if the transport layer protocol is TCP |
| 29 | UDP | Indicates if the transport layer protocol is UDP |
| 30 | DHCP | Indicates if the application layer protocol is DHCP |
| 31 | ARP | Indicates if the link layer protocol is ARP |
| 32 | ICMP | Indicates if the network layer protocol is ICMP |
| 33 | IPv | Indicates if the network layer protocol is IP |
| 34 | LLC | Indicates if the link layer protocol is LLC |
| 35 | Tot sum | Summation of packets lengths in flow |
| 36 | Min | Minimum packet length in the flow |
| 37 | Max | Maximumpacket length in the flow |
| 38 | AVG | Average packet length in the flow |
| 39 | Std | Standard deviation of packet length in the flow |
| 40 | Tot size | Packet's length |
| 41 | IAT | The time difference with the previous packet |
| 42 | Number | The number of packets in the flow |
| 43 | Magnitude | (Average of the lengths of incoming packets in the flow + average of the lengths of outgoing packets in the flow)$^{0.5}$ |
| 44 | Radius | (Variance of the lengths of incoming packets in the flow + variance of the lengths of outgoing packets in the flow)$^{0.5}$ |
| 45 | Covariance | Covariance of the lengths of incoming and outgoing packets |
| 46 | Variance | Variance of the lengths of incoming packets in the flow / variance of the lengths of outgoing packets in the flow |
| 47 | Weight | Number of incoming packets × Number of outgoing packets |

Figure 1: Extracted packet features in CICIoT2023

## 3.2 Step 2: Feature Selection

The CICIoT2023 dataset originally contains 47 features covering packet metadata, protocol flags, and statistical measures. However, not all features are equally informative or efficient for intrusion detection. Therefore 22 features are selected that are particularly useful for detecting malicious patterns, grouped as follows:

1. **Core Packet Metadata**

    - *Protocol Type (e.g., TCP, UDP, ICMP):* Critical for detecting protocol-specific attacks, such as TCP SYN floods or UDP storms.
    - *Header_Length:* Unusual header sizes can indicate malformed packets or exploits (e.g., buffer overflows).
    - *Tim_To_Live (TTL):* Variations in TTL may reveal routing anomalies or spoofing attempts.
    - *Rate:* Sudden spikes in packet rate often signify flooding attacks (e.g., DDoS).

2. **TCP Flags**

    - *fin_flag_number, syn_flag_number, rst_flag_number, psh_flag_number, ack_flag_number, ece_flag_number, cwr_flag_number:* These flags reveal connection states and anomalies. Excessive SYNs without ACKs, for instance, can indicate scanning, spoofing, or DoS attacks.

3. **Protocol Specific Indicators**

    - *TCP, UDP, ICMP, ARP, DNS, HTTP, HTTPS:* The presence or absence of these protocols helps distinguish normal from malicious behaviors (e.g., unexpected ARP in a sensor network, or HTTP usage from a device that normally should not use HTTP).

4. **Statistical Features**

    - *IAT (Inter Arrival Time):* Timing between packets can reveal bursty attack patterns versus steady, benign flows.
    - *Tot size (Total Size):* Oversized packets or anomalous sizes can suggest data exfiltration or exploit attempts.
    - *Avg (Average packet size):* Complements *Tot size* by providing an overall trend of packet sizes over time.

Focusing on these 22 features captures essential details about packet structure, protocol usage, and statistical signatures of normal versus attack traffic, reducing dimensionality while retaining high discriminative power.

### 3.3 Step 3: Model Selection

I evaluate five supervised learning models, chosen for their widespread use and complementary strengths:

1. **Logistic Regression:** A baseline model that is simple, interpretable, and relatively quick to train.
2. **BERT base:** Although typically used in NLP, BERT can be adapted for classification tasks given appropriate input representations.
3. **LightGBM:** A gradient boosting framework known for its efficiency and high performance on tabular data.
4. **Deep Neural Network (DNN):** Capable of modeling complex, non-linear relationships if sufficiently trained.
5. **Random Forest:** An ensemble of decision trees that often provides robust results with moderate tuning.

### 3.4 Step 4: Binary Classification

Finally, the traffic is classified(packets or flows) as either benign or attack. Tests indicate that a binary approach delivers higher accuracy compared to a more granular multi-class approach(tested with 8 and 24 classes). This simplification is especially relevant in real-world IoT environments, where the key question is whether traffic is safe or malicious, rather than the exact subtype of attack.

## 4 Model Implementation

This section outlines the specific implementation details for five different machine learning algorithms, focusing on how the *CICIoT2023* dataset was preprocessed and utilized in each case. After data transformation and model training, each approach is evaluated on standard metrics (accuracy, precision, recall, F1-score).

## 4.1 BERT-Base En Uncased

- **Textual Feature Transformation for BERT Initialization**: The raw packet features (e.g., `Protocol Type`, `Header_Length`) are concatenated into text strings through a `row_to_text` function. This enables a BERT classifier (`keras_hubmodelsBertClassifierfrom_preset ("bert_base_en", num_classes=2)`) with its pretrained 12 layer architecture to classify IoT traffic as benign or attack.

- **Selective Data Loading and Compilation**: A subset of 22 critical features is selectively loaded from multiple CSV files. The BERT model is compiled with `SparseCategoricalCrossentropy` (`from_logits=True`), an Adam optimizer (learning rate = $5 \times 10^{-5}$), and `accuracy` metric.

- **Label Encoding and Training Workflow**: The `Label` column is transformed into numeric values (0 for benign, 1 for attack) via a `label_to_num` function. Training uses a batch size of 32 for one epoch, with an 80/20 train-test split.

- **Dataset Splitting with Prediction and Evaluation**: After preprocessing, the trained BERT model infers test labels using `predict`, converting logits to binary classes. Performance is measured with accuracy, precision, recall, and F1-score.

**Model Evaluation (BERT-Base):**

- Accuracy: 0.9767
- Recall: 0.9765
- Precision: 0.9842
- F1-Score: 0.9803

## 4.2 Logistic Regression

- **Numerical Feature Selection and Model Initialization**: This approach loads 20 numerical packet features (e.g., `Header_Length`, `Rate`) from the CSV files, excluding `Protocol Type` to suit Logistic Regression's numerical input requirements.

- **Label Encoding and Training Process**: The `Label` column is converted from textual categories (`Benign`, `Attack`) to numeric labels (0, 1) using `label_to_num`, enabling binary classification. An 80% training subset is used to fit the model.
- **Dataset Splitting with Direct Feature Use**: After splitting into 80% training and 20% testing, the numerical features (e.g., `IAT`, `Tot size`) are directly used by Logistic Regression without any text transformation.
- **Prediction and Performance Evaluation**: The trained model predicts test labels via `predict`, followed by `accuracy_score` and `classification_report` to compute metrics.

**Model Evaluation (Logistic Regression):**

- Accuracy: 0.9890
- Recall: 0.8906
- Precision: 0.8641
- F1-Score: 0.8766

## 4.3 LightGBM (Ghourabi *et al.* [3])

- **Categorical Encoding with Optimized Model Initialization**: The `Protocol Type` field is one-hot encoded (`pd.get_dummies`), creating numerical binary columns. The LightGBM library then initializes a gradient-boosted decision tree model (`lgb.Dataset`).
- **Selective Feature Loading and Efficient Training**: 21 numerical features (plus `Label`) are loaded. LightGBM typically trains fastest among the models, running for 100 boosting rounds.
- **Label Transformation and Boosting Implementation**: The `Label` column is converted from textual (`Benign`, `Attack`) to numeric (0,1). LightGBM uses `objective='binary'` with histogram-based boosting and early stopping.

- **Direct Numerical Processing with Prediction and Evaluation**: Numerical features, split into training and testing sets, feed directly into LightGBM. Probabilities from `bst.predict` are thresholded at 0.5 to obtain binary labels.

**Model Evaluation (LightGBM):**

- Accuracy: 0.9906
- Recall: 0.9951
- Precision: 0.9953
- F1-Score: 0.9952

## 4.4 Random Forests

- **Categorical Encoding and Model Initialization**: `Protocol Type` is one-hot encoded (`pd.get_dummies`). A `RandomForestClassifier` with `n_estimators=100` and `max_depth=10` is then created.
- **Selective Feature Loading with Efficient Training**: 21 numerical features plus `Label` are loaded using `usecols`, followed by an 80/20 train-test split to train the model quickly.
- **Label Transformation and Handling Extreme Values**: The `Label` column is encoded (0,1). Inf or oversized values (e.g., in `Rate`) are capped at the float32 maximum to prevent overflow.
- **Direct Numerical Processing with Prediction and Evaluation**: Random Forest directly uses the numerical feature vectors; predictions come from `predict`, followed by standard scikit-learn metrics for evaluation.

**Model Evaluation (Random Forests):**

- Accuracy: 0.9901
- Recall: 0.9955
- Precision: 0.9944
- F1-Score: 0.9949

## 4.5 Neural Network (Chang *et al.* [8])

Five key points summarize this approach, adapted from the lightweight detection strategy proposed by Chang *et al.* for use in programmable switches:

- Uses exactly two hidden layers for a balance between high detection accuracy and lightweight computation.
- Each hidden layer has 128 neurons, aligning with Chang *et al.*'s guidelines for on-switch resource constraints.
- ReLU activation in the hidden layers captures non-linear features, with a single neuron output using sigmoid for binary classification (Benign vs. Attack).
- Standard hyperparameters (`batch_size=256`, `epochs=20`) offer a good trade-off between efficiency and convergence.
- The network is trained with binary crossentropy, suitable for two-class detection schemes.

**Model Evaluation (Neural Network):**

- Accuracy: 0.9913
- Recall: 0.9950
- Precision: 0.9961
- F1-Score: 0.9955

# 5 Evaluation Metrics

For each model, we split the dataset into training and testing sets (commonly 80%/20%). Training involves learning patterns from the labeled dataset, while testing validates model generalization. We measure performance with the following evaluation metrics:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

$$F1\text{-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

Where:

- TP = True Positives
- TN = True Negatives
- FP = False Positives
- FN = False Negatives

*Note:* We also track the number of rows per attack category to ensure balanced representation (or apply oversampling/undersampling if required).

# 6 Results

Evaluations of the tested models indicate that using a binary classification approach (benign vs. attack) both simplifies the decision boundary and generally achieves higher accuracy. Specifically:

- **LightGBM** tends to excel in handling large tabular data with many features, displaying strong recall (low false negatives).
- **Random Forest** often yields robust results with minimal tuning.

- **Neural Networks** can match or surpass traditional methods if carefully tuned (e.g., adjusting layer sizes, epochs, learning rate).
- **Logistic Regression** provides a strong baseline, interpretable coefficients, and relatively fast inference times.
- **BERT base** can leverage textual or packet content features when properly processed, although this approach is more complex and requires significant computing resources

All models are evaluated via the accuracy, precision, recall, and F1-score metrics described above, allowing thorough comparison of their strengths in distinguishing attacks from normal traffic.

# 7 Conclusion

This paper presented an approach to IoT anomaly detection using the CICIoT2023 dataset. By selecting 20 critical features, we achieve a lightweight yet effective method for identifying malicious traffic. Our preliminary experiments with five different models show promising results, reinforcing that binary classification can simplify detection and improve performance. Future work includes exploring additional feature-engineering techniques and evaluating more advanced deep learning architectures.

# References


[1] CICIoT2023 Dataset: `https://www.unb.ca/cic/datasets/index.html` (Accessed on: [date])

[2] Cisco Annual Internet Report—Cisco Annual Internet Report (2018–2023) White Paper. (n.d.). Cisco. Retrieved March 14, 2025, from https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[3] S. S. Roy, A. Mallik, R. Gulati, M. S. Obaidat and P. V. Krishna, "A deep learning based artificial neural network approach for intrusion detection", International Conference on Mathematics and Computing, pp. 44-53, 2017.

[4] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system", IEEE Access, vol. 7, pp. 41 525-41 550, 2019.

[5] P. Devan and N. Khare, "An efficient xgboost–dnn-based classification model for network intrusion detection system", Neural Computing and Applications, pp. 1-16, 2020.

[6] A. A. Hady, A. Ghubaish, T. Salman, D. Unal, and R. Jain, "Intrusion detection system for healthcare systems using medical and network data: A comparison study," IEEE Access, vol. 8, pp. 106576–106584, 2020.

[7] J. Dong Lee, H. Soung Cha, S. Rathore, and J. Hyuk Park, "M-IDM: A multi-classification based intrusion detection model in healthcare IoT," Comput., Mater. Continua, vol. 67, no. 2, pp. 1537–1553, 2021.

[8] H. -F. Chang, M. I. . -C. Wang, C. -H. Hung and C. H. . -P. Wen, "Enabling Malware Detection with Machine Learning on Programmable Switch," NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 2022, pp. 1-5, doi: 10.1109/NOMS54207.2022.9789939. keywords: Network intrusion detection;Detectors;Switches;Machine learning;Network security;Feature extraction;Malware.