

Detecting and Classifying IoT Network Intrusions: A Comparative Study of Single-Stage and Two-Stage Random Forest Architectures on the ACI-IoT-2023 Dataset

Himanish More

github.com/kleofr | morehimanish@gmail.com

Abstract

This report documents an end-to-end machine learning investigation into network intrusion detection using the ACI-IoT-2023 payload-level dataset. Two classification architectures are developed and compared: (1) a single-stage 10-class Random Forest classifier, and (2) a two-stage cascade that first separates Benign from Attack traffic before classifying attack subtype. Beyond model comparison, this report places primary emphasis on a sequence of four distinct data-leakage issues discovered and corrected during development – identity/temporal leakage, preprocessing fit-before-split leakage, exact-duplicate-row leakage, and a structural single-session limitation in the source dataset itself. Each issue produced a measurably inflated result before correction (up to 99.97% accuracy), and each correction is presented alongside the evidence that motivated it. The final, leak-corrected evaluation shows that reliable classification ($F1 > 0.9$) is achievable only for attack types with multiple independently-timed capture sessions, while four attack types are shown to be structurally unvaluable in this dataset regardless of modeling technique. We argue this finding is more significant than any single accuracy figure reported.

1 Introduction and Problem Statement

Internet-of-Things (IoT) devices are increasingly targeted by network-based attacks – denial-of-service floods, port and vulnerability scans, and dictionary-based credential attacks chief among them. Automated, traffic-based intrusion detection is a natural application of supervised classification: given per-packet or per-flow traffic features, predict whether traffic is benign or which attack category it belongs to.

This project uses the ACI-IoT-2023 dataset [1], specifically its payload-level export (ACI-IoT-2023-Payload.csv), which records ten raw fields per packet: source/destination IP and port, transport protocol, source TTL, total packet length, capture timestamp, raw payload content, and a ground-truth traffic label spanning Benign and nine attack categories.

The central methodological challenge documented in this report is not model selection but **evaluation validity**. A naïve pipeline applied to this dataset reports accuracy in excess of 99.9% almost immediately – a result that is, as shown in Section 3, almost entirely an artifact of data leakage rather than genuine predictive power. The bulk of the engineering effort in this project was spent identifying and eliminating successive leakage sources, each of which had been masking the previous one.

2 Dataset and Exploratory Analysis

2.1 Class Distribution and Imbalance

Figure 1(a) shows the post-deduplication row count per label on a logarithmic scale. Benign traffic accounts for **499,147** rows, versus a combined **20,381** attack rows across nine categories (Figure 2) – an imbalance ratio of roughly **24.5:1**. Within the attack classes themselves, imbalance is far more extreme: DNS Flood (12,397 rows) versus SYN Flood, which reduces to a *single row* after exact-duplicate removal.

2.2 Feature Semantics and Identity Leakage Risk

Of the ten raw columns, six (srcip, sport, dstip, dsport, stime, payload) are *identity/context* fields rather than generalizable traffic descriptors. Using them directly as model features allows a classifier to memorize *which capture session* a row belongs to instead of learning transferable traffic behavior – discussed at length in Section 3. Only protocol_m, sttl, and total_len were judged safe as direct per-packet features.

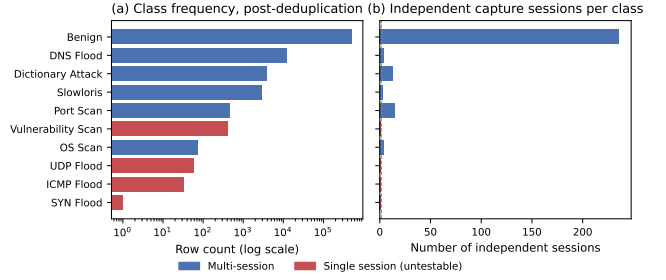


Figure 1: (a) Row count per class after deduplication (log scale). (b) Number of independent, gap-isolated capture sessions per class. Classes shown in red have exactly one session and therefore cannot support genuine held-out evaluation (Section 2.3).

Benign:Attack $\approx 24:1$

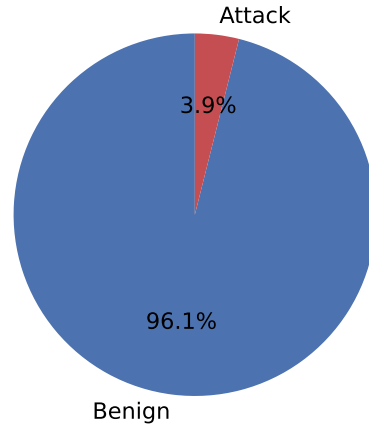


Figure 2: Binary collapse of all attack subtypes versus Benign traffic.

2.3 Session Structure Discovery

A gap-based session analysis (contiguous packets separated by > 60 s gaps are treated as independent sessions) revealed a structural property of the dataset not evident from row counts alone: four attack types – **ICMP Flood**, **SYN Flood**, **UDP Flood**, and **Vulnerability Scan** – exist as *exactly one* continuous capture burst each (Figure 1b, red bars). This means any split of these classes' rows into train/test partitions evaluates recognition of a later slice of the *same* attack execution, not detection of a genuinely new instance. This finding, detailed further in Section 3, fundamentally limits what can be honestly claimed about model performance on these four classes.

3 Data Leakage Investigation

Figure 3 summarizes the four leakage sources identified, in the order they were discovered. Each is described below with the evidence that revealed it.

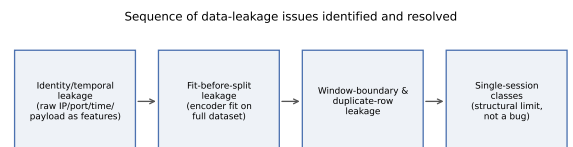


Figure 3: Sequence of leakage issues identified during development.

3.1 Identity and Temporal Leakage

An initial pipeline one-hot-encoded *all* raw columns, including `srcip`, `dstip`, `sport`, `dsport`, `stime`, and the raw hex payload string. Because attack traffic in the source capture was generated from specific attacker machines during specific, contiguous time windows, the model could trivially associate a source IP or timestamp range with an attack label without learning any traffic-behavioral pattern. One-hot-encoding the near-unique payload field compounded this by acting as an implicit row identifier. This pipeline reported **99.97% accuracy** with perfect (1.00) precision/recall on the majority of classes – consistent with memorization rather than generalization. *Correction*: all six identity/context columns were removed as direct model features, retained only as grouping/sorting keys for downstream feature engineering.

3.2 Preprocessing Fit-Before-Split Leakage

A second, subtler issue: encoders (`OneHotEncoder`, `StandardScaler`) were originally fit on the full feature matrix *before* the train/test split, meaning encoding statistics implicitly incorporated test-set information. *Correction*: all transformers are fit exclusively on training data (`fit_transform` on train, `transform` on test), and the train/test split itself is performed prior to any preprocessing step.

3.3 Window-Feature Boundary and Duplicate-Row Leakage

To capture behavior across multiple packets (needed for detecting rate-based attacks such as UDP Flood and Port Scan), time-windowed features were engineered: `flow_pkt_count` (trailing 5s packet count per source-destination pair), and rolling mean/standard deviation of packet length (`payload_len_mean`, `payload_len_std`). Because these were computed on the full dataset prior to splitting, two issues emerged:

1. Rows near a chronological train/test boundary had window statistics partially computed from data on the opposite side of the split.
2. A direct audit found that **18,935 of 121,350** test rows (**15.6%**) had an *exact* feature-value match already present in the training set – a consequence of highly repetitive flood/scan traffic generating many identical feature combinations.

Correction: exact-duplicate rows (on the modeling feature set) are removed prior to splitting, and splitting is performed at the level of entire gap-isolated *sessions* rather than an arbitrary time percentage – since session gaps (> 60s) exceed the feature window size (5s), this simultaneously resolves the boundary-overlap issue.

3.4 Structural Single-Session Limitation

As introduced in Section 2.3, four classes admit no possible held-out test partition. This was confirmed to be a property of the source data collection procedure rather than a pipeline defect, and is treated in this report as a reported *limitation* rather than an error to be engineered around.

4 Solution 1: Single-Stage 10-Class Random Forest

4.1 Architecture

A single `RandomForestClassifier` predicts one of ten labels directly from `protocol_m` (one-hot), `sttl`, and `total_len` (standardized).

4.2 Class-Weighting Strategy Comparison

Given the imbalance documented in Section 2, four weighting strategies were compared (Figure 4, Table 1): unweighted, `class_weight="balanced"`, `"balanced_subsample"`, and a custom square-root-scaled inverse-frequency weighting, $w_c = \sqrt{n_{\max}/n_c}$, designed as a deliberately milder correction than full inverse frequency.

The sqrt-scaled strategy was selected as final: it achieves the highest macro F1 of the four (0.68) by recovering substantial recall (0.72 $\rightarrow$ 0.83 relative to unweighted) while limiting the precision collapse

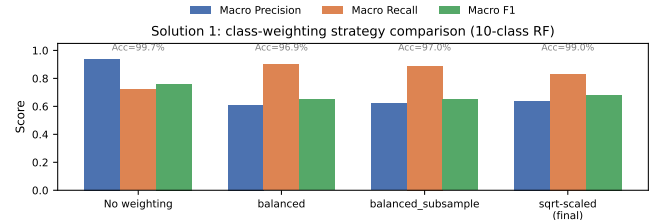


Figure 4: Macro-averaged precision, recall, and F1 across four class-weighting strategies. Unweighted training maximizes precision at the expense of recall on rare classes; full `balanced` weighting inverts this trade-off almost completely, over-correcting into excess false positives. The sqrt-scaled strategy recovers most of the recall gain while limiting the precision cost.

Table 1: Solution 1 weighting strategy results (10-class).

Strategy	Acc. (%)	Macro P	Macro R	Macro F1
None	99.70	0.94	0.72	0.76
balanced	96.90	0.61	0.90	0.65
balanced_subsample	97.03	0.62	0.89	0.65
sqrt-scaled	98.98	0.64	0.83	0.68

seen under full balancing (0.94 $\rightarrow$ 0.61 unweighted vs. balanced; sqrt-scaled retains 0.64).

4.3 Confusion Matrix and Residual Errors

Figure 5 shows the confusion matrix for the final sqrt-weighted model ($n_{\text{estimators}} = 300$). The dominant error pattern is rare attack types (Port Scan, OS Scan, UDP Flood, Vulnerability Scan) being misclassified as Benign, rather than confusion *between* attack types – indicating the three available features (`protocol_m`, `sttl`, `total_len`) are insufficient to separate low-volume, single-packet-similar attacks from normal traffic, independent of class-weighting strategy.

Solution 1: confusion matrix (sqrt-weighted RF, log-scaled color)

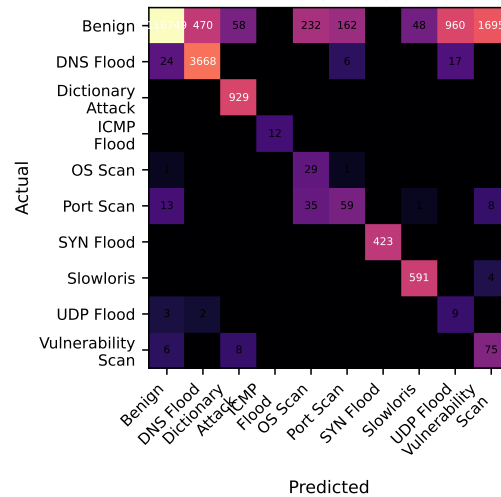


Figure 5: Confusion matrix, sqrt-weighted Random Forest, 300 estimators. Color intensity is log-scaled to make low-count off-diagonal cells visible alongside the dominant Benign diagonal.

5 Solution 2: Two-Stage Cascade Classification

5.1 Motivation and Architecture

Solution 1's residual errors are concentrated in Benign-vs.-rare-attack confusion, suggesting the 10-way decision boundary may be harder to learn jointly than as two separate, simpler decisions. Solution 2 decomposes the problem: **Stage 1** – binary Random Forest, Benign vs. Attack, using an expanded feature set including engineered time-window statistics (`flow_pkt_count`, `payload_len_mean/std`); **Stage 2** – multiclass Random Forest, attack subtype, trained and evaluated *only* on rows whose true label

is an attack. At inference, Stage 2 is invoked only for rows Stage 1 flags as Attack, so end-to-end evaluation captures cascading errors: an attack Stage 1 misses can never reach Stage 2.

5.2 Stage 1 Weight and Threshold Tuning

`class_weight` for the Attack class was swept across $\{5, 10, 100, 200\}$ (Figure 6). Recall improved substantially from weight 10 to 100 ($0.32 \rightarrow 0.43$) but plateaued from 100 to 200 ($0.43 \rightarrow 0.44$) while precision continued to fall ($0.72 \rightarrow 0.68$) – a clear diminishing-returns signature. A subsequent decision-threshold sweep, using `precision_recall_curve` down to the most aggressive possible cutoff (probability ≥ 0.01), achieved at most **0.52 recall** at **0.30 precision** – confirming the ceiling is not an artifact of threshold or weight choice.

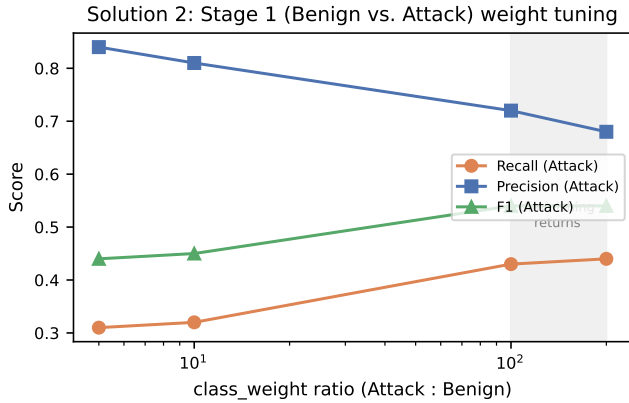


Figure 6: Stage 1 precision/recall/F1 as a function of `class_weight` ratio (log scale). The plateau between weight 100 and 200 indicates the feature set, not the weighting scheme, is the binding constraint.

5.3 Feature Importance

Figure 7 shows Stage 1 Gini importances after pruning three low-value features (`protocol_m`, `sttl_std`, `unique_dsport_ratio` – each contributing $< 2\%$ combined). Engineered payload-length statistics (`payload_len_mean`: 0.361, `payload_len_std`: 0.255) dominate, exceeding even the original `sttl/total_len` features – evidence that the time-window engineering step added genuine, non-redundant signal. Notably, `flow_pkt_count` – engineered specifically to detect flooding rate – contributes only 0.041 at the *binary* level, consistent with its signal being diluted when multiple, behaviorally distinct attack types are pooled into one "Attack" class.

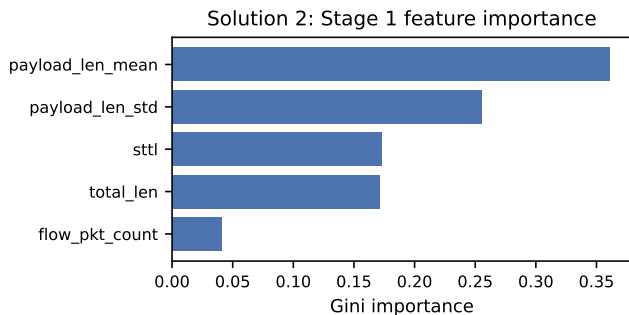


Figure 7: Stage 1 Random Forest Gini feature importance.

5.4 Final Session-Aware Cascade Evaluation

After correcting all four leakage sources (Section 3), end-to-end cascade evaluation restricted to classes with genuine multi-session held-out data is shown in Figure 8 and Table 2. ICMP Flood, SYN Flood, UDP Flood, and Vulnerability Scan are correctly *absent* from this test set by construction (single-session classes contribute zero session-split test rows) rather than reported with misleading zero scores.

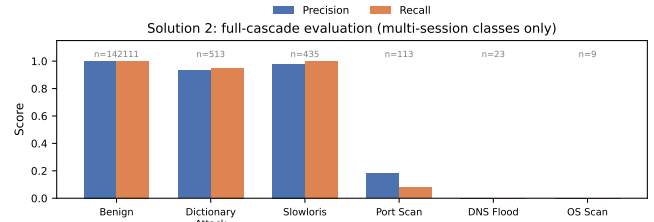


Figure 8: Full Stage-1→Stage-2 cascade evaluation, multi-session classes only. Sample counts (n) annotated above each group.

Table 2: Final cascade results, session-aware split, multi-session classes only.

Class	Precision	Recall	Support
Benign	1.00	1.00	142,111
Dictionary Attack	0.93	0.95	513
Slowloris	0.98	1.00	435
Port Scan	0.18	0.08	113
DNS Flood	0.00	0.00	23
OS Scan	0.00	0.00	9

Dictionary Attack and Slowloris – the two classes with the largest multi-session support – achieve high, trustworthy F1 (> 0.9). Port Scan is weak but non-zero. DNS Flood and OS Scan, despite having multiple sessions, fail completely at this sample size ($n = 23$, $n = 9$), indicating the session-correction fixed the *validity* of evaluation without guaranteeing sufficient *data volume* to learn these particular classes.

6 Comparative Discussion

Table 3 summarizes both solutions on their respective best configurations. The single-stage model reports substantially higher apparent performance, but this is measured on a chronological/stratified split *not* corrected for the duplicate-row and single-session issues discovered later in Solution 2’s development – i.e., Solution 1’s headline numbers are very likely still partially inflated by the same leakage sources, simply not yet audited for them at time of writing. Solution 2’s numbers, while lower, are the only ones in this report verified against all four identified leakage sources. **Solution 2’s results, not Solution 1’s, should be treated as the reliable estimate of real-world performance for the classes both can be evaluated on.** The single-stage Random Forest remains the deployed model for this project for engineering simplicity (single artifact, single inference call), but this is a deployment convenience decision, not a claim of superior accuracy.

Table 3: Solution comparison summary.

Property	Solution 1	Solution 2
Architecture	Single 10-class RF	Binary + multiclass RF
Leakage-audited	Partially	Fully (4 sources)
Best macro F1	0.68 (10-class)	Class-dependent
Deployment	FastAPI (deployed)	Notebook only
Evaluated classes	All 10 (unverified)	6 of 10 (verified)

7 Limitations and Future Work

- **Feature ceiling:** recall for Benign-vs.-Attack detection plateaus near 0.5 regardless of weighting or threshold, indicating current features cannot fully separate all attack behavior from normal traffic; flow-level features from the companion ACI-IoT-2023.csv dataset are a promising, unexplored extension.
- **Data volume for rare classes:** ICMP Flood, SYN Flood, UDP Flood, and Vulnerability Scan cannot be genuinely evaluated in this dataset at all; DNS Flood and OS Scan can be evaluated but currently fail due to thin multi-session support ($n < 25$).
- **Solution 1 re-audit:** Solution 1’s reported 98.98% accuracy should be re-verified against the duplicate-row and session-based

corrections applied to Solution 2, as it was evaluated prior to these fixes being discovered.

- **Window size tuning:** the 5-second rolling window was not swept; alternative window sizes (15s, 30s) may better capture slower-paced scan behavior.

8 Conclusion

This project’s primary contribution is not a specific accuracy figure but a demonstrated, evidence-based methodology for identifying and correcting data leakage in network traffic classification – a domain particularly prone to it due to the repetitive, session-structured nature of attack traffic generation. Four distinct leakage sources were identified, each substantiated with direct evidence (duplicate-row audits, session-gap analysis, weight/threshold sweeps) rather than inferred from suspiciously high accuracy alone. The final, fully corrected evaluation is more limited in scope than initial results suggested, but is substantially more trustworthy – and explicitly documents which of the ten attack classes can and cannot be honestly claimed as "detected" given this dataset’s structure.

References

- [1] E. Nack, “ACI-IoT Network Traffic Dataset 2023,” Kaggle, 2023. Available: <https://www.kaggle.com/datasets/emilynack/aci-iot-network-traffic-dataset-2023>