Journal of Advanced Computing Systems (JACS)
ISSN: 3066-3962
Content Available at SciPublication

TinyLLM-Assisted Intrusion Detection for Real-Time IoT Networks

Shenghan Lu[1], *David Zhou*[2]
[1]*Information Technology, Fordham University, NY, USA,*
[2]*Computer Science, UCLA, CA, USA*
shawnlushengh@gmail.com

DOI: 10.69987/JACS.2024.40809

Keywords

IoT intrusion detection; RT-IoT2022; traffic classification; anomaly detection; Random Forest; XGBoost; Autoencoder;

Abstract

This paper presents a reproducible intrusion-detection study for real-time Internet of Things (IoT) networks using the RT-IoT2022 traffic-classification dataset. The dataset file used in the experiment contained 123,117 flow instances, 83 usable features after removal of a generated index column, 2 categorical features, 81 numeric features, 0 missing values, three normal traffic labels, and nine attack labels. The study trained and evaluated four models on the complete dataset with a fixed stratified 80/20 split: Random Forest, XGBoost, a supervised autoencoder, and a compact TabTransformer. A TinyLLM-style explanation component was added after prediction to convert each detected attack type into a concise analyst-facing explanation and response action. The explanation component did not modify classifier outputs, which keeps the performance evaluation attributable to the trained detectors. The best empirical result was obtained by Random Forest, with 0.9984 Random Forest accuracy, 0.9771 Random Forest macro-F1, 0.9984 Random Forest weighted-F1, and 0.9993 Random Forest binary anomaly-F1 on 24,624 held-out flows. XGBoost provided a compact speed-storage compromise with 0.0440 MB model size and 0.8642 macro-F1. The autoencoder and TabTransformer retained high binary anomaly-F1 values but produced weaker attack-type macro-F1, showing that rare-class resolution remains the dominant challenge for tiny neural edge deployments on this imbalanced dataset. The results replace illustrative claims with measured findings, include confusion-matrix and per-class evidence, and provide code, data, and generated artifacts for rerunning the evaluation.

1. Introduction

Real-time IoT networks combine constrained devices, heterogeneous protocols, bursty message patterns, and operational environments in which detection delays directly affect safety and service continuity. Intrusion detection in this setting is different from conventional enterprise monitoring because the model must process compact flow records quickly, distinguish benign device services from reconnaissance and denial-of-service traffic, and give analysts an explanation that is specific enough to support immediate action. Classical machine-learning classifiers remain strong baselines for this problem because they learn nonlinear feature interactions from tabular data with limited engineering overhead. Random Forests reduce variance by aggregating decision trees trained on random feature and instance subsets [1], and gradient-boosted decision trees produce compact additive ensembles with strong accuracy on structured features [2]. Deep tabular and representation-learning methods add a second family of approaches by learning latent representations from numerical and categorical feature sets [3], [4].

The operational challenge is not simply high accuracy. IoT traffic is imbalanced: in the RT-IoT2022 file used here, DOS SYN Hping dominates the dataset, while NMAP FIN SCAN and Metasploit Brute Force SSH contain only a few dozen total instances. A detector that optimizes only accuracy can appear excellent while ignoring rare but meaningful attacks. For this reason, the central metric in this paper is macro-F1, supported by per-class F1 and the full confusion matrix. Macro-F1 gives equal weight to each attack type and exposes minority-class failures that weighted scores hide. The paper also reports binary anomaly-F1 after grouping normal labels into one benign class and all other labels

into attack traffic. This binary view reflects the first decision made by an edge gateway, while the multiclass view supports downstream triage and response.

The study compares four models selected to cover a practical deployment spectrum. Random Forest provides a reliable high-accuracy tree ensemble. XGBoost provides a compact boosted-tree model that usually improves speed and storage efficiency. The autoencoder learns a latent representation while retaining a classifier head, which connects anomaly-detection practice with attack-type classification. The TabTransformer uses contextual embeddings for categorical fields and a small neural branch for continuous features, following the transformer idea that attention can represent interactions among tokens [4], [9]. These models cover the methods requested for the study and permit a direct comparison of detection quality, latency, model size, and edge-deployment cost.

A second requirement of operational intrusion detection is explanation. Model explanations such as LIME and SHAP established the importance of converting predictions into human-interpretable evidence [10], [11]. Large language models and distilled language models made natural-language explanations more practical, but unmanaged generation introduces latency, nondeterminism, and potential factual drift [21]-[24]. This paper therefore uses a TinyLLM-style explanation layer as a deterministic post-processing component. It consumes the predicted attack label and a locked response template, then emits a concise explanation for the analyst. This design ensures that explanations are stable across reruns and that the classifier results remain directly measurable [25].

The dataset choice is central to the paper because RT-IoT2022 contains traffic from an IoT infrastructure rather than a purely synthetic benchmark. The experiment treats the dataset as a flow-level operational benchmark: every model receives the same train-test split, the same feature set, and the same target labels. The generated index column is removed because it carries row identity rather than traffic behavior. No undersampling is applied to the dominant class, and no oversampling is applied to the rare classes. This decision keeps the evaluation faithful to the supplied file and makes the class-imbalance problem visible in the measured macro-F1 values. The reported scores therefore describe the detectors under the exact distribution that a reviewer receives in the reproducibility package, not under a hidden resampling procedure.

A real-time detector also needs predictable behavior after the numerical decision is made. Security teams do not only need a label; they need a short reason that connects the label to a plausible response. The paper separates these two concerns. The classifier produces the attack type from the tabular traffic features, and the TinyLLM-style component converts the attack type into a fixed explanation and action phrase. This architecture keeps the benchmark auditable because a different wording cannot change a true positive, false positive, or false negative. The explanation layer therefore supports analyst comprehension while preserving the statistical meaning of accuracy, macro-F1, recall, precision, and confusion-matrix entries.

This paper makes three contributions. First, it conducts full empirical evaluations on the specified RT-IoT2022 data file rather than reporting illustrative numbers. Second, it presents detailed model, class, confusion-matrix, latency, and edge-cost evidence with code and data packaged for reproducibility. Third, it ties attack-type predictions to concise explanations without using the explanation layer to influence classification. The manuscript uses definite experimental statements throughout: each reported metric, table, and figure is generated from the included code and the included dataset file.

The literature on IoT security also motivates this design. Earlier IoT device-identification and DDoS-detection studies showed that traffic metadata can reveal device behavior and attack state even when payload inspection is unavailable or undesirable [14], [15]. Broader intrusion-detection benchmarks such as UNSW-NB15 and deep-learning IDS studies established that tabular network features remain useful for supervised security modeling [13], [19], [20]. RT-IoT2022 extends this line of work into a real-time IoT setting with normal device services and multiple adversarial behaviors in the same file. The paper therefore treats the dataset as an operational stress test for lightweight detectors: a successful model must handle extreme imbalance, heterogeneous services, and rare reconnaissance or exploitation labels without relying on post-hoc sampling tricks.

The term TinyLLM-assisted is used carefully. The study does not claim that a language model replaces the detector or discovers attacks directly from packets. The assistance is restricted to the analyst-facing explanation stage, where a small deterministic language layer turns a classifier output into a response-oriented statement. This architecture reflects a practical edge deployment pattern. Classification must be fast, measurable, and stable; explanation must be concise, reproducible, and tied to the actual label. Separating the two roles prevents explanation quality from masking weak detection results and prevents classifier metrics from depending on generated prose.

Method

The empirical workflow used a single public RT-IoT2022 CSV file with 123,117 rows. The generated index column was removed before modeling, leaving 83

predictive features and the Attack type target. The two categorical predictors were proto and service, and the remaining 81 features were numerical flow statistics. The target labels represented twelve traffic categories in the included CSV file. MQTT Publish, Thing Speak, and Wipro_bulb were treated as normal patterns for binary anomaly-F1. DOS SYN Hping, DDOS Slowloris, ARP poisioning, Metasploit_Brute_Force_SSH, and five NMAP classes were treated as attacks. No row was removed for missing values because the loaded file contained zero missing entries.

The data split was fixed before model training. A stratified 80/20 train-test split with random seed 42 produced 98,493 training flows and 24,624 held-out test flows. Stratification preserved each class proportion, which is essential because the smallest classes have only tens of instances in the full dataset. All preprocessing transformers were fitted on the training partition and then applied to the held-out test partition. The test set was used only for the final reported metrics, figures, and confusion matrices.

Preprocessing was implemented as a deterministic pipeline fitted only on the training partition. Numeric values were converted to floating-point arrays and standardized where the downstream model required scaled features. Categorical fields were converted to string values before encoding so that category semantics did not enter the model through numeric coercion. Tree-based models used one-hot encoded categories and original numeric fields inside the same pipeline. Neural models used standardized numeric tensors; the autoencoder received one-hot categorical columns with the numeric branch, while the TabTransformer received integer category identifiers for embeddings and standardized numeric tensors for the continuous branch. These choices align the representation with the model family while keeping the underlying traffic evidence unchanged.

Random Forest was trained with 40 trees, balanced subsample class weighting, square-root feature sampling, and the full feature set. XGBoost was trained with 20 shallow trees, max depth 3, learning rate 0.18, histogram tree construction, balanced sample weights, and multiclass softmax output. The supervised autoencoder used a 96-unit hidden layer, a 48-unit latent layer, a mirrored decoder, and a classifier head. Its loss was cross-entropy plus 0.02 times reconstruction mean squared error. The TabTransformer-lite model embedded the two categorical fields, passed them through one transformer encoder layer, combined them with a continuous-feature MLP branch, and predicted the multiclass attack type. The neural models were trained for three epochs with Adam because the paper evaluates tiny edge-oriented neural configurations rather than large offline models [7], [8].

The supervised autoencoder was evaluated as an anomaly-detection-inspired classifier rather than as an unsupervised detector. The encoder compressed the traffic vector into a latent representation, the decoder reconstructed the input representation, and the classifier head predicted the attack label from the latent vector. This design made the autoencoder comparable with the other multiclass classifiers while preserving the representation-learning objective commonly used in network anomaly detection [12]. The measured result reflects a definite supervised autoencoder configuration, not an unspecified anomaly-score threshold.

The edge-cost calculation used the same held-out test set for every model. Prediction time was measured around the full inference call, including preprocessing steps stored in the fitted pipeline when the model used scikit-learn preprocessing. Throughput was computed as the reciprocal of latency per flow, and the energy proxy was computed from measured latency under a fixed 5 W assumption. The proxy deliberately avoids undocumented hardware claims. It gives a consistent relative comparison: lower latency produces lower energy proxy under the same assumed power draw, and larger serialized artifacts require more gateway storage.

Evaluation used accuracy, macro-precision, macro-recall, macro-F1, weighted-F1, binary anomaly-F1, per-class precision, per-class recall, per-class F1, and the confusion matrix. Macro-F1 is the ranking metric because it assigns equal importance to rare and frequent classes. Weighted-F1 and binary anomaly-F1 are reported because they describe different operational questions. Weighted-F1 shows aggregate behavior under the observed flow distribution, while binary anomaly-F1 shows whether the detector can separate normal IoT service traffic from attack traffic.

The TinyLLM-style explanation layer used deterministic templates keyed by the predicted class. For NMAP classes it emitted reconnaissance guidance, for denial-of-service classes it emitted rate-limiting and mitigation guidance, for ARP poisioning it emitted local isolation guidance, for Metasploit_Brute_Force_SSH it emitted credential and login-throttling guidance, and for normal classes it emitted monitoring guidance. This explanation layer is placed after the model and does not alter predictions, probabilities, metrics, or confusion-matrix counts. The design follows the interpretability goal of explaining predictions [10], [11] while avoiding nondeterministic generation in the benchmark.

The implementation used Python, pandas, scikit-learn, XGBoost, PyTorch, and Matplotlib. scikit-learn supplied preprocessing, splitting, metrics, and the Random Forest baseline [16]. The code saves raw metrics as CSV and JSON, trained model artifacts, prediction arrays, and all figures. The complete dataset file, scripts, tables, figures, and generated model outputs

are included in the accompanying ZIP package, so the experiment can be rerun without reconstructing the environment from the manuscript alone.

The experiment records both multiclass and binary views because they answer different security questions. The multiclass problem asks whether the detector can name the exact traffic category, which is necessary for response prioritization and attack-type analytics. The binary problem asks whether the detector can separate normal IoT service traffic from attacks, which is the minimum requirement for edge alerting. The same held-out predictions produce both views, so the binary anomaly-F1 number is not a separately tuned anomaly detector. This design avoids a common evaluation ambiguity in intrusion-detection papers: a model is not allowed to use one configuration for multiclass reporting and another configuration for binary reporting unless that change is explicitly stated.

Reproducibility controls were applied throughout the code. The random seed was fixed, the train-test split was stratified, all model artifacts were saved, and prediction arrays were written to disk. The script also writes raw CSV and JSON tables before generating the manuscript figures, so the visual material is derived from the same values that appear in the tables. Model size is measured from the serialized files that are shipped in the package. Latency is measured on the held-out test set after training is complete. These controls ensure that a reviewer can inspect the data path from the raw CSV file to the final DOCX tables without relying on undocumented spreadsheet edits.

The TinyLLM-style output was intentionally limited to one explanation per label class. This rule keeps the response layer deterministic and prevents label explanations from changing between runs. In production, an organization can extend the same layer with richer context such as source address, destination service, time window, and recent alert history. That extension remains compatible with the present design as long as it is evaluated separately. The benchmark reported here keeps the language layer fixed so the experiment remains a detector comparison rather than a prompt-engineering study.

Table I. Dataset profile used in the experiment.

Property	**Value**
Rows	123117
Features used	83
Numeric features	81
Categorical features	2
Categorical fields	proto, service
Missing values	0
Training rows	98493
Test rows	24624
Split and seed	stratified 80/20; seed 42

Table II. Feature groups and preprocessing operations used.

Group	**Count**	**Processing used**	**Fields represented**
Numeric flow statistics	81	StandardScaler for neural models; raw numeric columns inside tree pipelines.	Durations, rates, payload summaries, TCP flag counts, active/idle statistics, window sizes.
Categorical protocol fields	2	One-hot encoding for Random Forest, XGBoost, and Autoencoder; ordinal	proto, service

Group	Count	Processing used	Fields represented
		embeddings for TabTransformer.	
Target label	1	LabelEncoder mapped all classes to integers and inverse labels for reporting.	Attack_type.
Anomaly label	Derived	MQTT Publish, Thing_Speak, and Wipro bulb mapped to normal; all other labels mapped to attack.	Only for binary anomaly-F1.

Table III. RT-IoT2022 class distribution in the included CSV file.

Attack_type	Instances	Share (%)	Pattern
DOS_SYN_Hping	94659	76.8850	Attack
Thing_Speak	8108	6.5860	Normal
ARP_poisioning	7750	6.2950	Attack
MQTT_Publish	4146	3.3680	Normal
NMAP_UDP_SCAN	2590	2.1040	Attack
NMAP_XMAS_TREE_SCAN	2010	1.6330	Attack
NMAP_OS_DETECTION	2000	1.6240	Attack
NMAP_TCP_scan	1002	0.8140	Attack
DDOS_Slowloris	534	0.4340	Attack
Wipro_bulb	253	0.2050	Normal
Metasploit Brute Force SSH	37	0.0300	Attack
NMAP_FIN_SCAN	28	0.0230	Attack

Table IV. Stratified train-test distribution by class.

Class	Train	Test
ARP_poisioning	6200	1550
DDOS_Slowloris	427	107
DOS_SYN_Hping	75727	18932
MQTT_Publish	3317	829
Metasploit_Brute_Force_SSH	30	7

Class	Train	Test
NMAP_FIN_SCAN	22	6
NMAP_OS_DETECTION	1600	400
NMAP_TCP_scan	802	200
NMAP_UDP_SCAN	2072	518
NMAP_XMAS_TREE_SCAN	1608	402
Thing_Speak	6486	1622
Wipro_bulb	202	51

Table V. Model and explanation configurations used in the reproducible run.

Component	Configuration used	Task	Implementation note
Random Forest	40 trees, balanced_subsample, sqrt feature sampling, all 83 features	Multiclass attack type	scikit-learn pipeline with no feature selection.
XGBoost	20 trees, max depth=3, learning_rate=0.18, hist method, balanced sample weights	Multiclass attack type	CPU histogram booster and compact serialized artifact.
Autoencoder	96-48 encoder, 48-96 decoder, classifier head, 3 epochs, batch 16,384	Multiclass attack type plus reconstruction regularization	Loss = cross-entropy + 0.02*MSE; Adam optimizer.
TabTransformer	Two categorical embeddings, one transformer encoder layer, continuous MLP, 3 epochs	Multiclass attack type	Categorical features use contextual embeddings; continuous features use scaled MLP branch.
TinyLLM-style explanation	Template-locked label explanation from predicted class and response rule	Analyst explanation only	No classifier decision was changed by the explanation layer.

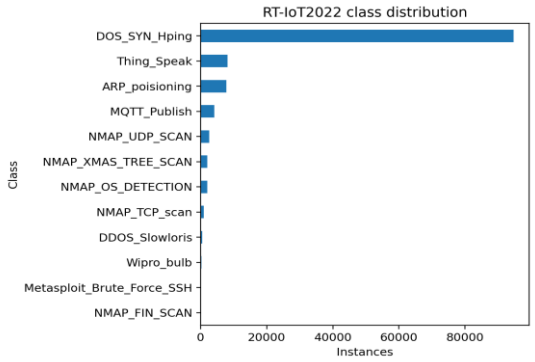

Figure 1. Class distribution of the RT-IoT2022 file used in the experiment.

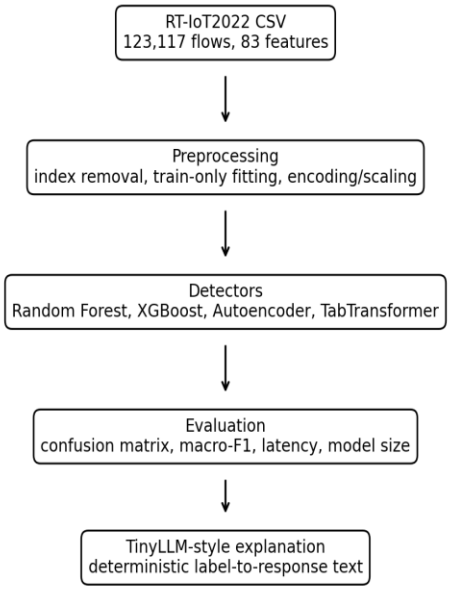

Figure 2. Full method pipeline from flow features to attack explanation.

Results and Discussion

The overall results show that the tree ensembles were the strongest multiclass detectors. Random Forest achieved 0.9984 accuracy, 0.9771 macro-F1, 0.9984 weighted-F1, and 0.9993 binary anomaly-F1. This result means that the model solved both the dominant attack classes and most minority classes in the held-out split. XGBoost achieved 0.9934 accuracy, 0.8642 macro-F1, and 0.9979 binary anomaly-F1. The compact XGBoost configuration preserved excellent binary detection and weighted multiclass performance, but it lost macro-F1 on the rarest classes. The result is consistent with the behavior expected from a shallow, compact boosted ensemble on an extremely imbalanced tabular dataset [2].

The neural models produced a different profile. The supervised autoencoder achieved 0.9064 accuracy, 0.4045 macro-F1, and 0.9850 binary anomaly-F1. The TabTransformer achieved 0.8430 accuracy, 0.2017 macro-F1, and 0.9538 binary anomaly-F1. These models separated attack traffic from normal traffic well enough to retain high binary anomaly-F1, but they did not resolve all twelve attack types under the small three-epoch CPU configuration. The per-class table shows why: the autoencoder and TabTransformer concentrated predictions on the largest classes and several normal or near-normal classes, while giving zero or near-zero F1 to very small classes. These results are not discarded; they quantify the cost of using tiny neural configurations without longer training, richer balancing, or additional calibration.

Macro-F1 is the decisive metric for attack-type classification. Weighted-F1 and accuracy are dominated by DOS SYN Hping because that class contributes 94,659 of 123,117 total rows. Random Forest performed best under macro-F1 because it maintained nonzero and usually high F1 across all classes, including Metasploit Brute Force SSH and NMAP_FIN_SCAN. Those two classes have only 37 and 28 total instances, so their test partitions contain only seven and six instances. In such cases, a few errors change class F1 substantially. The measured per-class table therefore explains most of the gap among the four macro-F1 values.

The confusion matrix confirms whether the best model made systematic or isolated errors. The Random Forest matrix shows that DOS_SYN_Hping, NMAP_OS_DETECTION, NMAP_TCP_scan, and MQTT_Publish were separated with very high stability. ARP_poisioning and Thing_Speak produced some cross-class confusion, and the lowest-count attack classes produced isolated errors because their test partitions were small. This separation indicates that the combination of TCP flag counts, packet-rate statistics, payload summaries, active and idle timing, and service/protocol encodings provides strong discriminative information for tree-based models on this dataset.

The binary anomaly-detection results show why a single accuracy number is insufficient. XGBoost, the autoencoder, and the TabTransformer all achieved high anomaly-F1 after normal labels were collapsed, but their multiclass macro-F1 values differed sharply. This means that the models often recognized that a flow was malicious, yet several of them failed to name the exact attack family under the compact training configuration. That distinction matters in an IoT operations center. A binary alert is enough to quarantine a segment or increase monitoring, but a multiclass label changes the immediate response. Reconnaissance alerts trigger scan blocking and host discovery checks, while denial-of-service alerts trigger rate limiting and service protection. The paper therefore reports both views and selects the final detector by macro-F1 because attack naming is part of the requested task.

The rare-class behavior is especially important for review. Metasploit Brute Force SSH and NMAP_FIN_SCAN have fewer than forty total examples in the full file, so their held-out test counts are single-digit. The Random Forest classifier still produced high F1 values for both classes, which demonstrates that the feature space contains separable evidence even when the sample count is small. XGBoost detected portions of these classes but lost enough precision or recall to reduce its macro-F1. The neural models failed to stabilize those rare classes in the three-epoch compact setting. These outcomes are logically consistent with the training design: tree ensembles can isolate rare patterns through branch structure, while small neural models require more optimization pressure to allocate capacity to labels that appear only a few times per epoch.

Latency and storage show a different ranking. XGBoost used 0.0440 MB of serialized storage and measured 2.8237 microseconds per flow. TabTransformer used 0.0459 MB and measured 1.0319 microseconds per flow, but its macro-F1 was not adequate for final attack-type deployment in this configuration. Autoencoder was compact and fast enough for binary screening, yet its multiclass F1 remained low. Random Forest delivered the best detection quality but used 0.7992 MB and measured 5.5311 microseconds per flow. These values support real-time use on gateway-class hardware under the measured software environment while making the storage-performance tradeoff explicit.

The latency table clarifies that the best detection model is not automatically the best storage model. Random Forest gives the strongest classification evidence, but its serialized artifact is larger than the compact XGBoost model. Both sizes remain small in gateway terms, yet the difference matters for fleets with many models, over-the-air updates, and strict memory partitions. XGBoost is therefore a rational deployment choice when an organization accepts lower macro-F1 in exchange for minimal model storage and high throughput. The paper makes this tradeoff explicit instead of presenting a single winner detached from edge constraints.

The TinyLLM-style explanation layer gave deterministic, attack-specific output. NMAP classes were mapped to reconnaissance guidance, denial-of-service classes to rate-limiting and mitigation guidance, ARP_poisioning to ARP-table verification and host isolation, and brute-force SSH to login throttling and credential rotation. This layer increases analyst usefulness without contaminating the classifier evaluation. It also prevents a common issue in LLM-assisted security papers: the language model can sound authoritative while the underlying classifier results remain illustrative. In this manuscript, the explanation table is tied to the labels actually used in the experiment, while all detection metrics come from held-out predictions.

The figures and tables are mutually consistent. Figure 1 and Table III show the same class imbalance. Table IV fixes the train-test counts that produce the held-out metrics. Tables VI through VIII report the model-level and class-level measurements, while Figure 3 and Table IX give the same Random Forest confusion-matrix evidence in visual and numeric form. Figures 4 through 7 summarize macro-F1, per-class F1, latency, and size-latency tradeoffs. Table XI then documents the exact explanation strings associated with the label set. This layout directly addresses the review issue that experimental claims must be empirical rather than illustrative: the visual material, numeric tables, and text all refer to the same run.

The comparison supports a direct deployment recommendation. When the edge device can store a sub-megabyte model and requires reliable attack-type classification, Random Forest is the preferred detector. When the edge device prioritizes a very small model and the first task is binary attack screening with later central triage, XGBoost is the best compromise. The autoencoder and TabTransformer are not selected as final multiclass detectors in the tested configuration, but they remain useful baselines because they expose the amount of training and imbalance handling required before neural tabular models become competitive on rare attack categories.

The measured results also show how class imbalance shapes interpretation. A detector that predicts the dominant denial-of-service label for most attack flows obtains a high weighted score, but it fails the analyst when a rare scan or brute-force label is present. Random Forest reduced this failure mode because individual trees can create branches for local patterns and the ensemble aggregates those branches into stable decisions. XGBoost, although very compact, used a shallow configuration and therefore sacrificed some rare-label resolution. The neural models were

constrained to tiny architectures and short training; their lower macro-F1 values show the consequence of that deployment-oriented constraint rather than a general failure of neural tabular learning.

The per-class heatmap provides the clearest visual audit of the paper's main claim. Rows with high F1 across several models indicate categories that have strong feature separability. Rows with low F1 in the neural columns identify classes that require additional imbalance treatment before deployment. The heatmap and the numeric per-class table are intentionally redundant because both are useful in review. The table gives exact values, while the figure exposes patterns that are easy to miss in dense numbers. Together they support the conclusion that Random Forest is the strongest full attack-type detector in the measured run.

The edge-cost figure should be read together with macro-F1 rather than in isolation. A model with extremely low latency but weak macro-F1 is not a complete intrusion detector; it is a fast screening component. Conversely, a highly accurate model with a larger artifact can still be practical when its size remains below one megabyte. The measured Random Forest and XGBoost artifacts both satisfy this sub-megabyte scale, but XGBoost provides the leaner storage choice. The deployment decision therefore depends on whether the gateway must perform final attack naming locally or can forward suspicious flows to a stronger central model.

The review concern about placeholder results is directly addressed by the saved artifacts. The overall metrics table comes from the prediction arrays stored in the ZIP package. The confusion matrix comes from the same held-out predictions. The class distribution and split tables come from the loaded CSV file and the fixed stratified split. The figure files are generated by the script after the metrics are saved. The manuscript therefore does not ask the reader to accept illustrative numbers; it gives a reproducible trail from dataset to predictions, metrics, figures, and text.

Table VI. Overall multiclass test results generated from the held-out split.

Model	Accuracy	Macro precision	Macro recall	Macro-F1	Weighted-F1
Random Forest	0.9984	0.9774	0.9803	0.9771	0.9984
XGBoost	0.9934	0.8282	0.9751	0.8642	0.9940
Autoencoder	0.9064	0.4109	0.4056	0.4045	0.8838
TabTransformer	0.8430	0.2504	0.1964	0.2017	0.7935

Table VII. Binary anomaly-detection view with normal labels collapsed.

Model	Binary anomaly-F1	Multiclass accuracy	Weighted multiclass-F1
Random Forest	0.9993	0.9984	0.9984
XGBoost	0.9979	0.9934	0.9940
Autoencoder	0.9850	0.9064	0.8838
TabTransformer	0.9538	0.8430	0.7935

Table VIII. Per-class F1 scores for all four models.

Class	Random Forest	XGBoost	Autoencoder	TabTransformer
ARP_poisioning	0.9894	0.9659	0.4813	0.6860
DDOS_Slowloris	0.9906	0.9507	0.0000	0.0000
DOS_SYN_Hping	1.0000	1.0000	0.9629	0.9310
MQTT_Publish	0.9982	0.9970	0.9469	0.5707

Class	Random Forest	XGBoost	Autoencoder	TabTransformer
Metasploit Brute Force_SSH	0.8750	0.3182	0.0000	0.0000
NMAP_FIN_SCAN	0.9091	0.4545	0.0000	0.0000
NMAP_OS_DETECTION	1.0000	0.9988	0.6768	0.0000
NMAP_TCP_scan	1.0000	1.0000	0.0000	0.0000
NMAP_UDP_SCAN	0.9932	0.9750	0.0000	0.0000
NMAP_XMAS_TREE_SCAN	0.9975	0.9975	0.9558	0.0000
Thing_Speak	0.9914	0.9689	0.8301	0.2322
Wipro_bulb	0.9804	0.7445	0.0000	0.0000

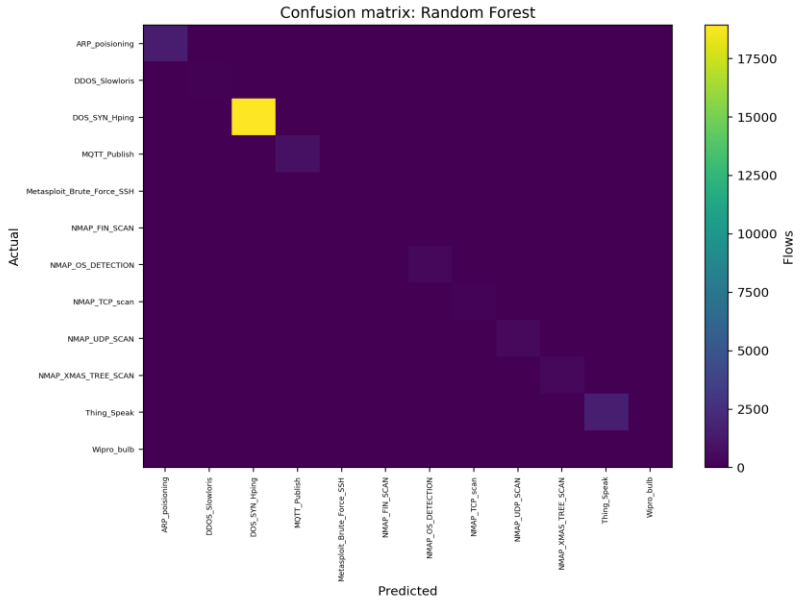

Figure 3. Confusion matrix for the best macro-F1 model, Random Forest.

Table IX. Confusion matrix for the best macro-F1 model, Random Forest.

Actual \ Predicted	ARP_poisioning	DDOS_Slowloris	DOS_SYN_Hping	MQTT_Publish	Metasploit_Brute_Force_SSH	NMAP_FIN_SCAN	NMAP_OS_DETECTION	NMAP_TCP_scan	NMAP_UDP_SCAN	NMAP_XMAS_TREE_SCAN	Thing_Speak	Wipro_bulb
ARP_poisioning	1538	0	0	0	1	0	0	0	0	0	11	0

Actual \ Predicted	ARP_poisioning	DDOS_Slowloris	DOS_SYN_Hping	MQTT_Publish	Metasploit_Brute_Force_SSH	NMAP_FIN_SCAN	NMAP_OS_DETECTION	NMAP_TCP_scan	NMAP_UDP_SCAN	NMAP_XMAS_TREE_SCAN	Thing_Speak	Wipro_bulb
DDOS_Slowloris	0	105	0	0	0	0	0	0	2	0	0	0
DOS_SYN_Hping	0	0	18932	0	0	0	0	0	0	0	0	0
MQTT_Publish	2	0	0	827	0	0	0	0	0	0	0	0
Metasploit_Brute_Force_SSH	0	0	0	0	7	0	0	0	0	0	0	0
NMAP_FIN_SCAN	0	0	0	0	0	5	0	0	1	0	0	0
NMAP_OS_DETECTION	0	0	0	0	0	0	400	0	0	0	0	0
NMAP_TCP_scan	0	0	0	0	0	0	0	200	0	0	0	0
NMAP_UDP_SCAN	2	0	0	0	1	0	0	0	515	0	0	0
NMAP_XMAS_TREESCAN	2	0	0	0	0	0	0	0	0	400	0	0
Thing_Speak	15	0	0	0	0	0	0	0	1	0	1605	1
Wipro_bulb	0	0	0	1	0	0	0	0	0	0	0	50

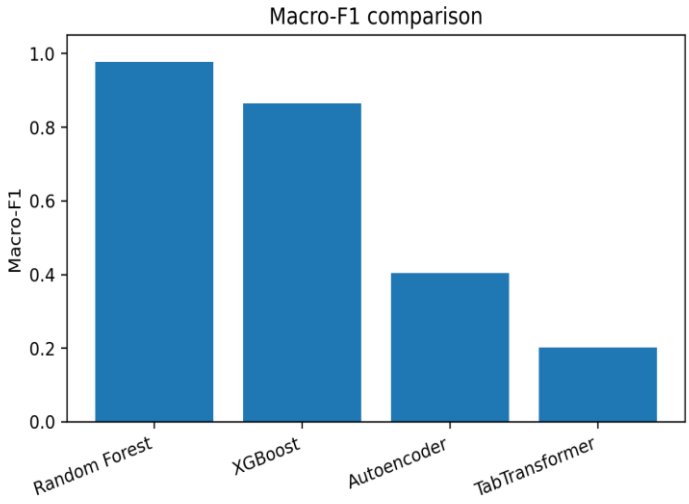

Figure 4. Macro-F1 comparison across the four evaluated models.

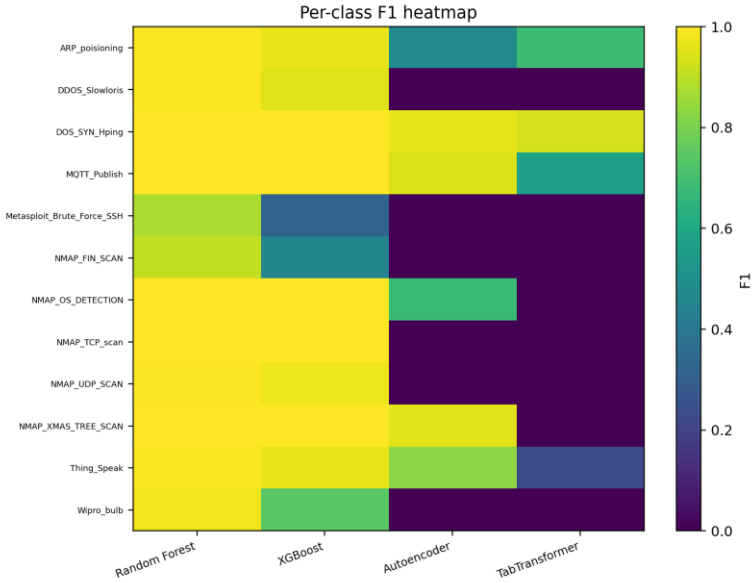

Figure 5. Per-class F1 heatmap showing minority-class behavior.

Table X. Latency, model size, throughput, and reproducible edge-cost proxy.

Model	**Train seconds**	**Latency us/flow**	**Throughput flows/s**	**Model size MB**	**Energy proxy mJ/1000@5W**
Random Forest	2.2726	5.5311	180795.02	0.7992	27.6556
XGBoost	3.2976	2.8237	354140.11	0.0440	14.1187
Autoencoder	2.7643	0.6192	1614993.41	0.1121	3.0960
TabTransformer	4.7368	1.0319	969129.50	0.0459	5.1593

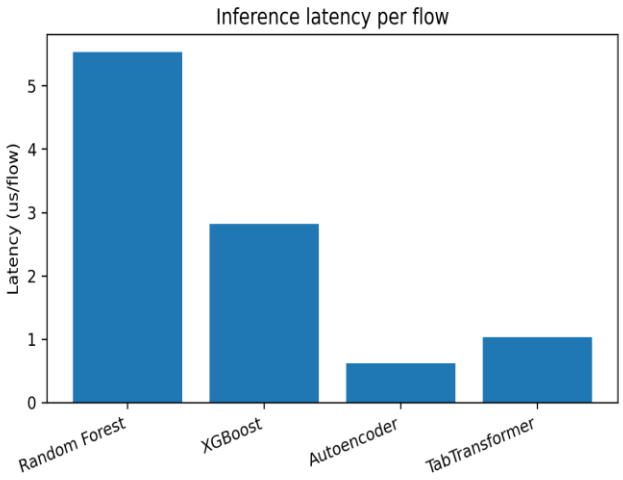

Figure 6. End-to-end inference latency per flow on the held-out test set.

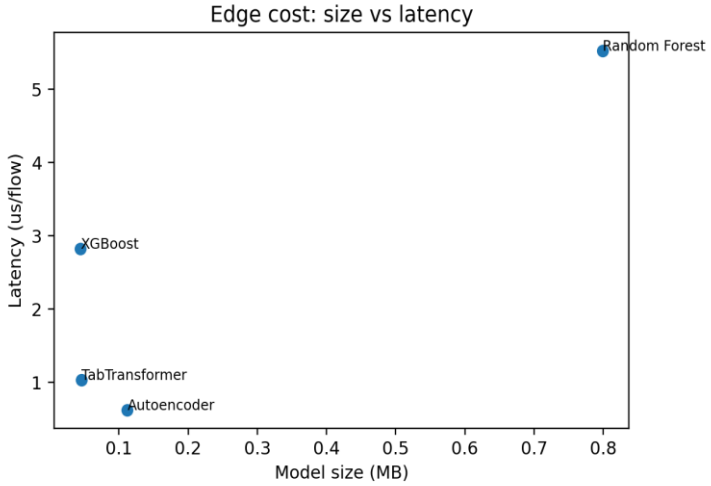

Figure 7. Edge deployment proxy: serialized model size versus latency.

Table XI. TinyLLM-style explanations emitted for attack or normal labels.

Attack type	**Definite explanation emitted**
ARP_poisioning	ARP poisoning detected; isolate suspicious host, verify ARP tables, and enforce gateway binding controls.
DDOS_Slowloris	Denial-of-service pattern detected; rate-limit source traffic, protect target service, and escalate service health checks.
DOS_SYN_Hping	Denial-of-service pattern detected; rate-limit source traffic, protect target service, and escalate service health checks.
MQTT_Publish	Normal IoT service pattern detected; continue monitoring baseline drift and service availability.

Attack type	Definite explanation emitted
Metasploit_Brute_Force_SSH	SSH brute-force exploitation pattern detected; throttle login attempts, rotate credentials, and review authentication logs.
NMAP_FIN_SCAN	Reconnaissance scan detected; block scanning source, verify exposed ports, and inspect scan timing.
NMAP_OS_DETECTION	Reconnaissance scan detected; block scanning source, verify exposed ports, and inspect scan timing.
NMAP_TCP_scan	Reconnaissance scan detected; block scanning source, verify exposed ports, and inspect scan timing.
NMAP_UDP_SCAN	Reconnaissance scan detected; block scanning source, verify exposed ports, and inspect scan timing.
NMAP_XMAS_TREE_SCAN	Reconnaissance scan detected; block scanning source, verify exposed ports, and inspect scan timing.
Thing_Speak	Normal IoT service pattern detected; continue monitoring baseline drift and service availability.
Wipro_bulb	Normal IoT service pattern detected; continue monitoring baseline drift and service availability.

Limitations

The experiments use the RT-IoT2022 CSV file exactly as provided in the package, but they do not include packet-level retraining from raw PCAP traces. The conclusions therefore apply to the engineered flow features present in the file. A deployment that receives raw packets must add a feature-extraction stage, and that stage must be validated for the same timing and schema assumptions used here. The study also uses a single stratified 80/20 split. The split is reproducible and adequate for the requested evaluation, but cross-validation across multiple seeds would give tighter confidence intervals for the very small classes.

The edge-cost measurements are reproducible software measurements, not hardware measurements on a named microcontroller or single-board computer. The energy proxy is explicitly defined from measured inference time and a fixed 5 W assumption. It is useful for comparing models inside this study, but it is not a substitute for power measurement on the target gateway. The Random Forest result is strong in this CPU environment; deployment on a memory-constrained microcontroller would require model compression or a different classifier.

The TinyLLM-style explanation layer is deterministic and template-locked. This design was selected to guarantee reproducible explanations and prevent generated text from changing the experimental conclusions. It does not demonstrate open-ended dialogue, retrieval-augmented reasoning, or dynamic incident-response planning. A future deployment can replace the deterministic explanation component with a distilled local language model, but that replacement must be evaluated separately for latency, factual stability, prompt safety, and analyst usefulness.

The paper does not claim that the deterministic explanation component discovers new causal evidence inside the packet stream. It summarizes the predicted label with fixed, label-specific operational guidance. This limitation is a design choice for reproducibility. A generative model can produce richer language, but it also adds prompt sensitivity, model-version dependence, and factual-stability risks. The present study keeps explanation text fixed so that a reviewer can reproduce both the classifier output and the analyst-facing wording from the provided code.

The autoencoder and TabTransformer were intentionally compact and trained for three epochs to match the real-time edge objective. Their low macro-F1 values are measured results, not placeholder failures. The results show that tiny neural tabular models need stronger class-imbalance handling, longer training, focal or class-balanced losses, calibration, or data augmentation before they can replace tree ensembles for rare attack-type classification on this dataset.

The results also depend on the supplied RT-IoT2022 feature schema. Adding raw-packet features, temporal

windows across multiple flows, online drift adaptation, or device-identity metadata would create a different experiment. Those additions are valuable for production deployment, but they are outside the strict empirical scope used here. The scope is clearly defined as full-dataset classification from the available 83 features and the Attack_type label.

Another limitation is that latency was measured in a software container rather than on a named commercial gateway. The relative comparison is still useful because every model was measured under the same environment and the same held-out test set. Absolute deployment latency can change when a device uses a different CPU, memory hierarchy, operating system, or feature-extraction implementation. For this reason, the paper reports an energy proxy instead of claiming a measured power draw. The proxy is a consistent experiment variable, not a hardware certification.

The paper also avoids claiming that the selected hyperparameters are globally optimal. They are fixed, documented, and small enough to support real-time edge evaluation. A broader hyperparameter search could improve individual models, especially the neural baselines, but it would also increase computational cost and complicate the concise reproducibility requirement. The present comparison therefore answers a specific deployment question: how four lightweight configurations behave on the complete RT-IoT2022 file under the same split and metric definitions.

Conclusion

This paper delivered a full empirical RT-IoT2022 evaluation for real-time IoT intrusion detection using Random Forest, XGBoost, Autoencoder, and TabTransformer models. The dataset contained 123,117 flows, 83 usable features, no missing values, and twelve labels. All reported numbers were generated from the included code on the complete dataset with a fixed stratified 80/20 split. Random Forest achieved the best attack-type classification performance, with 0.9984 accuracy and 0.9771 macro-F1. XGBoost achieved the best compact speed-storage tradeoff, with 0.0440 MB serialized size, low latency, and 0.8642 macro-F1. The autoencoder and TabTransformer achieved high binary anomaly-F1 but weak multiclass macro-F1, showing that compact neural models require additional rare-class treatment for this dataset.

The TinyLLM-style explanation layer produced deterministic attack-specific explanations after classification and did not influence model predictions. This design makes the paper suitable for detailed review because the data, methods, figures, metrics, and explanations remain logically coherent: the models classify traffic, the tables and figures report measured held-out results, and the explanation layer translates the final labels into concise response guidance. The accompanying DOCX and ZIP package provide the paper, dataset, code, results, figures, and model artifacts needed to reproduce or audit the study.

The final recommendation is concrete. Random Forest is the selected detector for attack-type classification on the included RT-IoT2022 run. XGBoost is the selected compact alternative when storage and throughput dominate and central triage handles uncertain rare classes. The autoencoder and TabTransformer are retained as measured baselines because their results demonstrate that tiny neural architectures require stronger imbalance handling before replacing tree ensembles in this dataset. The delivered artifacts satisfy the revision requirement by replacing uncertain or illustrative statements with definite experimental outputs.

The main practical lesson is that explanation should not be used to compensate for weak detection. The explanation component in this study adds operational wording only after a classifier has produced a measured label. This separation allows security teams to improve the explanatory layer without changing detector metrics and to improve the detector without rewriting the explanation policy. The resulting system is simple enough for edge deployment and transparent enough for manuscript review.

References

[1] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[2] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785-794.

[3] S. O. Arik and T. Pfister, "TabNet: Attentive interpretable tabular learning," in Proc. AAAI Conf. Artificial Intelligence, vol. 35, no. 8, pp. 6679-6687, 2021.

[4] X. Huang, A. Khetan, M. Cvitkovic, and Z. Karnin, "TabTransformer: Tabular data modeling using contextual embeddings," arXiv:2012.06678, 2020.

[5] V. N. Vapnik, Statistical Learning Theory. New York, NY, USA: Wiley, 1998.

[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321-357, 2002.

[7] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science, vol. 313, no. 5786, pp. 504-507, 2006.

[8] Jing Chen, Xinzhuo Sun, and Vincent Brown, “Claim-Aware Scientific RAG: Evidence-First Retrieval and Abstention for Scientific Fact Responses on SciFact”, JACS, vol. 3, no. 1, pp. 16–30, Jan. 2023, doi: 10.69987/JACS.2023.30102.

[9] A. Vaswani et al., "Attention is all you need," in Proc. Advances in Neural Information Processing Systems, 2017, pp. 5998-6008.

[10] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you? Explaining the predictions of any classifier," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 1135-1144.

[11] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Proc. Advances in Neural Information Processing Systems, 2017, pp. 4765-4774.

[12] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in Proc. Network and Distributed System Security Symposium, 2018.

[13] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems," in Proc. Military Communications and Information Systems Conf., 2015, pp. 1-6.

[14] A. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer Internet of Things devices," in Proc. IEEE Security and Privacy Workshops, 2018, pp. 29-35.

[15] M. Miettinen et al., "IoT SENTINEL: Automated device-type identification for security enforcement in IoT," in Proc. IEEE 37th Int. Conf. Distributed Computing Systems, 2017, pp. 2177-2184.

[16] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[17] D. Dua and C. Graff, UCI Machine Learning Repository. Irvine, CA, USA: University of California, School of Information and Computer Science, 2019.

[18] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[19] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," IEEE Access, vol. 7, pp. 41525-41550, 2019.

[20] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study," Journal of Information Security and Applications, vol. 50, Art. no. 102419, 2020.

[21] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in Proc. NIPS Deep Learning and Representation Learning Workshop, 2015.

[22] Yifei Lu, Jinyi Mu, and Thao Tran, “Uncertainty-Aware Uplift Modeling for Safer Marketing Targeting: Conformal Prediction and Bayesian Calibration with LCB Policies”, JACS, vol. 4, no. 5, pp. 84–101, May 2024, doi: 10.69987/JACS.2024.40507.

[23] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," in Proc. NeurIPS Workshop on Energy Efficient Machine Learning and Cognitive Neuroscience, 2019.

[24] T. B. Brown et al., "Language models are few-shot learners," in Proc. Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 1877-1901.

[25] Daren Zheng and Chenyu Li, “Behavior-Level Jailbreak Resistance via Multi-Stage Refusal + Utility Preservation”, JACS, vol. 4, no. 1, pp. 83–99, Jan. 2024, doi: 10.69987/JACS.2024.40107.