Journal of Communication Sciences and Information Technology (JCSIT)
An International Journal

A Comparative Analysis of Logistic Regression and Random Forest for Intrusion Detection Systems: A Reproducible, Deployment-Oriented Study

Ahmed T. Shawky*
*El Madina Higher Institute of Administration and Technology, Egypt
Correspondence: ahtaisser@gmail.com

Abstract
Intrusion Detection Systems (IDS) must deliver high detection rates with controllable false alarms under class imbalance, non-stationarity, and strict latency constraints. We present a rigorous, end-to-end comparison of **Logistic Regression (LR)** and **Random Forest (RF)** for IDS, emphasizing reproducibility, statistical validity, and deployability. Using standard benchmark datasets (NSL-KDD for tabular network connections; BoT-IoT and TON_IoT for modern IoT traffic), we build a unified pipeline covering feature preprocessing, class-imbalance mitigation, hyperparameter tuning, threshold calibration, and uncertainty-aware evaluation. Results show that **RF consistently achieves higher recall and F1 on attack classes**, especially under heterogeneous traffic, while **LR offers superior interpretability and competitive precision** when calibrated. We quantify operating regimes where LR is preferable (auditable environments, tight latency, scarce features) and where RF dominates (nonlinear patterns, richer features). We release a full protocol—metrics, statistical tests, ablations, and deployment guidelines—to enable reproducible benchmarking and practical adoption.

Keywords: Intrusion Detection Systems; Network Security; Class Imbalance; Logistic Regression; Random Forest; Threshold Calibration; ROC/PR Analysis; Explainability; Reproducibility.

1. Introduction
The dramatic growth of interconnected systems has expanded the attack surface of enterprise, cloud, and IoT environments. Intrusion Detection Systems (IDS) remain a cornerstone of cyber defense but face chronic challenges: non-stationary traffic distributions, severe class imbalance, and costly false alarms that can overwhelm Security Operations Centers (SOCs). Machine learning (ML) methods—particularly **Logistic Regression (LR)** and **Random Forest (RF)**—are frequently adopted for tabular network telemetry due to their robustness, speed, and maturity.

While prior work has compared LR and RF, many studies lack (i) a unified preprocessing and calibration protocol, (ii) comprehensive imbalance handling, (iii) strong statistical testing, and (iv) actionable guidance for field deployment. This paper addresses those gaps through a rigorous, fully specified pipeline, dataset triage across legacy and modern benchmarks, and operational recommendations grounded in measured trade-offs between **recall vs. precision**, **latency vs. accuracy**, and **interpretability vs. complexity**.

Contributions.

1. A reproducible LR/RF benchmarking pipeline spanning preprocessing, class-imbalance mitigation, cross-validation, threshold calibration, and uncertainty-aware reporting.
2. A principled evaluation across NSL-KDD and contemporary IoT datasets (BoT-IoT, TON_IoT), with statistical significance testing and error analysis.
3. Deployment guidance (thresholding, alert routing, model monitoring, and

drift handling) and an explainability toolkit suitable for SOC handoff.

4. A public protocol and artifact checklist to facilitate repeatable comparisons and industrial adoption.

In the digital age, the proliferation of interconnected systems and devices has led to an exponential increase in cyber threats. Intrusion Detection Systems (IDS) have emerged as a critical component in safeguarding networks against unauthorized access, malicious activities, and data breaches. IDS are designed to monitor network traffic, identify suspicious patterns, and alert administrators to potential threats. However, the effectiveness of IDS largely depends on the underlying algorithms used for detection.

Machine learning (ML) has revolutionized the field of cybersecurity by enabling IDS to learn from historical data and adapt to evolving threats. Among the various ML algorithms, **Logistic Regression (LR)** and **Random Forest (RF)** have gained prominence due to their simplicity, interpretability, and robustness. Logistic Regression is a statistical model used for binary classification, offering probabilistic insights into intrusion events. Random Forest, on the other hand, is an ensemble learning method that aggregates multiple decision trees to enhance prediction accuracy and resilience against overfitting.

This research aims to conduct a comparative analysis of LR and RF in the context of IDS, evaluating their performance across various metrics such as accuracy, precision, recall, and false alarm rate. The study leverages benchmark datasets like BoT-IoT and TON-IoT to assess the efficacy of these algorithms in detecting intrusions in real-world scenarios.

Classical supervised learning remains competitive for IDS on structured telemetry. **LR** offers calibrated probabilities and transparent coefficients, supporting governance and post-hoc auditing. **RF**—an ensemble of decision trees with randomized subspace selection—captures non-linear interactions and is resilient to outliers. Empirical meta-analyses show RF often outperforms linear baselines on complex tabular data, though at the cost of reduced interpretability. Benchmark datasets include **NSL-KDD** (improving upon KDD'99 to reduce redundancy), **BoT-IoT** (IoT botnet behaviors), and **TON_IoT** (realistic multi-source telemetry). Beyond accuracy, modern IDS studies stress **false alarm rate**, **class imbalance**, **cost-sensitive learning**, and **operational calibration** (ROC/PR analysis and threshold selection). We extend this line by unifying these elements in a single, end-to-end protocol with deployment-oriented guidance.

2. Literature Review

Machine learning has become central to modern intrusion detection, with Logistic Regression (LR) and Random Forest (RF) among the most frequently examined baselines and production candidates. A broad stream of comparative studies consistently reports that RF attains strong detection rates across heterogeneous traffic, while LR remains valuable as a transparent, well-calibrated reference—especially when latency, governance, or feature scarcity are binding constraints. Large-scale benchmarks across domains corroborate this pattern, showing that RF tends to dominate when nonlinearity and interaction effects are pronounced, whereas LR provides

competitive precision and auditability with modest computational cost and straightforward probability outputs (e.g., calibration-friendly scores) (Couronné et al., 2018).

Within IoT and IIoT settings, where telemetry is high-volume and attack distributions are non-stationary, RF has repeatedly emerged as a high-recall choice. Recent evaluations on BoT-IoT and related collections found RF superior to several classical models across multiple metrics, including accuracy and F1, with LR serving as a robust baseline for probability-quality and interpretability (Eid et al., 2024; Zolanvari et al., 2021; Song et al., 2022). In vehicular networks (IoV), both RF and LR have been integrated into hybrid and multitiered IDS architectures to address complex spatiotemporal dynamics and protocol diversity; RF typically underpins high-capacity detectors, while LR supports fast, auditable decision layers (Yang et al., 2019; Yang et al., 2021; Meghana Kiran et al., 2024). Beyond raw performance, studies emphasize the importance of dataset-specific preprocessing (e.g., NetFlow/Zeek field engineering), strict train/validation/test separation, and careful threshold selection to align detection with operational costs.

A second theme is ensemble learning and hybridization to balance recall, precision, and false-alarm control under class imbalance. Chalichalamala et al. (2023) proposed a Logistic Regression Ensemble Classifier that combines AdaBoost and RF to leverage complementary decision boundaries; their results on BoT-IoT demonstrate the efficacy of stacking linear and tree-based components for robust detection. Parallel efforts combine RF with feature selectors, cost-sensitive learning, or imbalance-aware sampling (e.g., Gini-impurity weighting and borderline-SMOTE variants) to improve minority-class sensitivity without destabilizing false-positive rates (Disha & Waheed, 2023; Koc et al., 2020; Roopadevi et al., 2020). These approaches typically report gains for rare attack families (e.g., R2L/U2R), where pure discriminative training can underfit due to data scarcity.

A third strand targets scalability and data realism. Sarhan et al. (2023) demonstrated the practicality of NetFlow-based feature sets across BoT-IoT and TON-IoT, highlighting pipeline choices (encoding, aggregation windows, and categorical handling) that preserve throughput while retaining discriminative power. Comparative works in industrial IoT (IIoT) echo these findings, noting that RF's embedded feature subsampling and robustness to outliers make it suitable for high-dimensional, partially noisy telemetry, while LR's linear structure remains advantageous when features are carefully curated or when strict explainability is mandated (Eid et al., 2024; Zolanvari et al., 2021; Arya, 2022; Jain & Srihari, 2024).

A fourth body of research addresses generalization to evolving threats. Transfer learning and semi-supervised strategies have been explored to bolster zero-day detection, often pairing RF with representation learning or label-efficient training to mitigate the brittleness of purely supervised pipelines (Rodríguez et al., 2023; Zhang et al., 2023). Hybrid data-mining schemes and improved probabilistic baselines (e.g., enhanced Naïve Bayes) continue to serve as useful controls or lightweight detectors in streaming contexts, but tree-based ensembles tend to retain the edge in recall for novel behaviors, particularly when enriched with up-to-date features and periodic recalibration (Farid et al., 2021; Koc et al., 2020).

Finally, several survey and framework papers converge on methodological best practices that are directly relevant to LR/RF assessments. WJARR (2022) emphasized

aligning algorithm choice with operational constraints (latency budgets, analyst capacity, and acceptable false-alarm rates), while Jain and Srihari (2024) highlighted binary LR as a reliable baseline for comparative testing due to its transparency and ease of calibration. More recent comparative analyses of decision trees, KNN, LR, and RF reiterate LR's interpretability and decision trees' computational efficiency, with RF providing the most consistent accuracy–recall trade-off across mixed traffic conditions (Shukla & Kumar, 2025; Zhang & Zulkernine, 2022). Taken together, these studies underscore the complementary strengths of LR and RF: LR offers governance-ready, probability-calibrated outputs for auditable deployments, whereas RF delivers higher recall and robustness in complex, imbalanced, and evolving intrusion landscapes. Despite this progress, a fully standardized, deployment-oriented comparison—covering preprocessing parity, imbalance mitigation, calibration, threshold selection against cost functions, and uncertainty-aware reporting—remains necessary to translate benchmark gains into reliable, real-world IDS performance across diverse environments.

3. Problem Statement

We evaluate LR and RF for binary (Normal vs. Attack) and multiclass (e.g., DoS, Probe, R2L, U2R) intrusion detection under:

- **Imbalanced labels** (rare attacks),
- **Nonlinear interactions** among categorical and numeric features,
- **Operational constraints** (low-latency scoring, explainability for SOCs), and
- **Generalization** from legacy to modern IoT contexts.

We seek to answer: (Q1) Which model best balances recall and false alarms? (Q2) How much do calibration and class-weighting help? (Q3) What deployment settings (thresholds, alert tiers) optimize SOC value? (Q4) What risks and failure modes remain?

We adopt a dataset triad to cover legacy and modern patterns:

- **NSL-KDD** (connections with numeric + categorical features): reduced redundancy vs. KDD'99; supports binary and family-level labels.
- **BoT-IoT** (IoT botnet traffic): large-scale, diverse attack profiles (DoS/DDoS, keylogging, data exfiltration).
- **TON_IoT** (telemetry & network): multi-source data (sensors, logs, network captures), enabling realistic, heterogeneous evaluation.

Preprocessing parity is enforced across datasets: consistent encoders, scalers, and label mappings; strict separation of train/test splits; and no leakage across folds.

Despite the advancements in IDS technologies, several challenges persist:

- **High False Alarm Rates**: Many IDS suffer from excessive false positives, leading to alert fatigue and reduced trust in the system.
- **Imbalanced Datasets**: Intrusion data is often skewed, with normal traffic dominating the dataset. This imbalance affects the learning capability of ML models.
- **Scalability and Adaptability**: IDS must handle large volumes of data and adapt to new attack vectors, which is challenging for traditional models.
- **Interpretability vs. Accuracy Trade-off**: While LR offers interpretability, it may lack the accuracy of more complex models like RF. Conversely,

RF provides high accuracy but is less transparent.

This study addresses these issues by comparing LR and RF in terms of their ability to detect intrusions effectively, handle imbalanced data, and provide actionable insights for cybersecurity professionals.

4. Datasets & Objectives

We adopt a dataset triad to cover legacy and modern patterns:

- **NSL-KDD** (connections with numeric + categorical features): reduced redundancy vs. KDD'99; supports binary and family-level labels.
- **BoT-IoT** (IoT botnet traffic): large-scale, diverse attack profiles (DoS/DDoS, keylogging, data exfiltration).
- **TON_IoT** (telemetry & network): multi-source data (sensors, logs, network captures), enabling realistic, heterogeneous evaluation.

Preprocessing parity is enforced across datasets: consistent encoders, scalers, and label mappings; strict separation of train/test splits; and no leakage across folds.

The primary objectives of this research are:

1. To evaluate the performance of Logistic Regression and Random Forest in detecting network intrusions.
2. To analyze their behavior on imbalanced datasets and assess their robustness.
3. To compare their interpretability, scalability, and computational efficiency.
4. To provide recommendations for selecting appropriate algorithms based on IDS requirements.

Significance of the Study

This study contributes to the field of cybersecurity by:

- Providing empirical evidence on the effectiveness of LR and RF in IDS.
- Offering insights into algorithm selection for different network environments.
- Enhancing the understanding of ML-based IDS design and deployment.

By bridging the gap between theoretical models and practical applications, this research aims to support the development of more secure and resilient network infrastructures.

5. Research Methodology

This section outlines the systematic approach adopted to conduct a comparative analysis of Logistic Regression and Random Forest algorithms in the context of Intrusion Detection Systems (IDS). The methodology encompasses data collection, preprocessing, model implementation, evaluation metrics, and experimental setup

5.1 Logistic Regression (LR)

For a feature vector x∈Rdx, LR models Pr(y=1\mid x) = \sigma(w^\top x + b) where σ is the sigmoid. We use **L2-regularized** LR with class weights to mitigate imbalance. For multiclass, we adopt one-vs-rest or multinomial LR depending on dataset characteristics. Post-training **Platt scaling** or isotonic regression may be applied for calibration.

5.2 Random Forest (RF)

RF aggregates TTT decision trees trained on bootstrapped samples and feature subspaces; predictions are majority vote (classification) with optional probability estimates via class frequency in leaves. We tune trees' depth, leaf size, and feature subsampling to balance bias/variance and latency.

5.3 Class-Imbalance Mitigation

We compare **class weights**, **SMOTE** (and borderline variants), **undersampling**, and **ensemble balancing** (e.g., balanced RF). We emphasize threshold tuning on validation PR curves to match operational cost functions.

5.4 Calibration and Thresholding

We report both **score-distribution** diagnostics and **calibration curves**. Thresholds are set to optimize (i) macro F1, (ii) attack recall @ acceptable false-positive rate (FPR), and (iii) cost-sensitive utility tailored to SOC capacity.

5.5 Explainability and Analyst Handoff

For LR, we interpret standardized coefficients and class-conditional odds. For RF, we report **permutation importance**, **Gini importance**, and optional **SHAP** analyses for local case explanations. Outputs are designed to feed SOC triage with **top contributing features** and **confidence**.

I. Research Design

The study employs a **quantitative experimental design** to evaluate and compare the performance of two machine learning algorithms—Logistic Regression (LR) and Random Forest (RF)—in detecting network intrusions. The design is structured to ensure reproducibility, statistical validity, and relevance to real-world IDS applications.

II. Dataset Selection

To ensure a comprehensive evaluation, the study utilizes publicly available benchmark datasets commonly used in IDS research:

- **BoT-IoT Dataset**: Contains simulated IoT network traffic with various attack types, including DoS, DDoS, reconnaissance, and data exfiltration.
- **TON-IoT Dataset**: Offers telemetry and network data from IoT devices, including normal and malicious traffic.

These datasets are selected for their diversity, volume, and relevance to modern network environments.

III. Data Preprocessing

Preprocessing is a critical step to enhance model performance and ensure data quality. The following procedures are applied:

- **Data Cleaning**: Removal of missing values, duplicates, and irrelevant features.
- **Feature Selection**: Use of correlation analysis and domain knowledge to select the most informative features.
- **Normalization/Standardization**: Scaling features to ensure uniformity, especially important for LR.
- **Label Encoding**: Conversion of categorical labels into numerical format for model compatibility.
- **Handling Imbalanced Data**: Techniques such as SMOTE (Synthetic Minority Over-sampling Technique) are applied to address class imbalance.

IV. Model Implementation

Two machine learning models are implemented using Python and relevant libraries (e.g., Scikit-learn):

- **Logistic Regression (LR)**: A linear model used for binary classification. It estimates the probability of an event occurring based on input features.
- **Random Forest (RF)**: An ensemble learning method that constructs multiple decision trees and aggregates their outputs to improve accuracy and reduce overfitting.

Hyperparameter tuning is performed using grid search and cross-validation to optimize model performance.

V. Evaluation Metrics

To assess and compare the models, the following performance metrics are used:

- **Accuracy**: Proportion of correctly classified instances.
- **Precision**: Ratio of true positives to all predicted positives.
- **Recall (Sensitivity)**: Ratio of true positives to all actual positives.
- **F1-Score**: Harmonic mean of precision and recall.
- **False Alarm Rate (FAR)**: Proportion of normal traffic incorrectly classified as malicious.
- **Area Under the ROC Curve (AUC-ROC)**: Measures the trade-off between true positive rate and false positive rate.

These metrics provide a holistic view of each model's effectiveness in intrusion detection.

6. Experimental Setup

Environment. Python 3.x; scikit-learn, pandas, numpy, matplotlib; optional SHAP for explainability.
Hardware. Commodity CPU (e.g., Intel i7, 16 GB RAM).
Splits. Train/validation/test with stratification; 5× repeated stratified k-fold where appropriate.
Tuning. Grid/random search over regularization (LR), class weights, and RF hyperparameters (n_estimators, max_depth, min_samples_leaf, max_features). fR
Reporting. Mean ± 95% CI over repeats; paired **Wilcoxon signed-rank tests** for LR vs. RF on core metrics; calibration error (ECE), AUROC, AUPRC
The experiments are conducted in a controlled environment with the following specifications:

- **Programming Language**: Python 3.x
- **Libraries**: Scikit-learn, Pandas, NumPy, Matplotlib, Seaborn
- **Hardware**: Intel Core i7 processor, 16GB RAM
- **Software**: Jupyter Notebook, Anaconda

Each model is trained and tested using an 80/20 train-test split, and results are averaged over multiple runs to ensure consistency.

VII. Comparative Analysis

The final step involves a detailed comparison of LR and RF based on the evaluation metrics. Statistical tests such as paired t-tests or Wilcoxon signed-rank tests may be used to determine the significance of performance differences. The analysis also considers factors like interpretability, computational efficiency, and scalability.

What we built: A machine-learning pipeline that ingests NSL-KDD network connection records, preprocesses features (categorical → one-hot, numeric → scaled), trains two classifiers (Logistic Regression and Random Forest) in **both binary** (Normal vs Attack) and **multiclass (attack family)** modes and produces **12+ visualizations** and a textual results report.

- **Primary role:** intrusion detection (IDS) — determines whether a network record is normal or an attack and categorize attack type.
- **Important observed metric:** Logistic Regression (binary) *precision Normal 0.65, recall Normal 0.93, f1 Normal 0.76; precision Attack 0.92, recall Attack 0.62, f1 Attack 0.74; accuracy*

0.75. This implies the model rarely mislabels normal records as attacks (high Normal recall), and predictions labelled Attack are usually correct (high Attack precision) but many real attacks are missed (Attack recall 0.62).

1 — What the algorithm does (role) and capabilities Role

- **Primary:** automated detection of anomalous/malicious network connections (IDS).
- **Secondary:** coarse triage — assign detected attacks to family labels (DoS, Probe, R2L, U2R), produce feature-level insight for analysts (which services/flags/fields are predictive).

Capabilities

- **Binary classification:** Normal vs Attack (suitable for real-time alerting).
- **Multiclass classification:** Identify attack families (helpful for prioritization and incident response).
- **Feature analysis & visualization:** identify top services/flags targeted, distribution of durations and bytes.
- **Model comparison:** compare a linear model (LogisticRegression) vs a tree ensemble (Random Forest).
- **Artifact generation:** saved models/encoders, confusion matrices, plots, and a textual report.

2 — High-level pipeline

1. **Read dataset files** (KDDTrain+.txt, KDDTest+.txt).
2. **Map labels → families** (DoS, Probe, R2L, U2R, Other).
3. **Binary label creation**: Normal → 0, Attack → 1.
4. **Feature selection**: categorical (protocol type, service, flag) and numeric columns.
5. **Preprocessing**:
   - OneHotEncoder (handle unknown='ignore') for categorical features.
   - StandardAero () for numeric features.
   - Combine using stack (to allow sparse outputs).
6. **Label encoding** of families via Label Encoder for multiclass training.
7. **Train models**:
   - Logistic Regression (baseline, interpretable, fast).
   - Random Forest Classifier (nonlinear patterns, handles interactions).
   - Models trained separately for *binary* and *multiclass* tasks.
8. **Evaluation**:
   - classification report (precision, recall, f1) and confusion matrices.
9. **Visualizations** — 12+ plots saved to figures/.
10. **Save textual results** to a results file.

3 — Libraries used and why

- **Python standard libs**: os, wget — file presence and optional download convenience.
- **pandas** — tabular data loading and manipulation. Chosen for robust CSV parsing and convenience of value_counts, filtering, and grouping.
- **numpy** — numeric arrays, used by scikit-learn and matplotlib.
- **scikit-learn**:
  - ✓ OneHotEncoder — to transform categorical features into numeric vectors. handle_unknown='ignore' avoids errors when test contains unseen categories (common in NSL-KDD).

- ✓ StandardAero — normalize numeric features so linear models and tree models behave more predictably.
- ✓ LabelEncoder — converts family names to numeric labels for multiclass.
- ✓ LogisticRegression — baseline, calibration friendly, fast for prototyping and interpretable coefficients.
- ✓ RandomForestClassifier — effective, handles non-linearities and interactions; little feature engineering required.
- ✓ classification_report, confusion_matrix — standard metrics for classification.

- **scipy.sparse.hstack** — to combine numeric dense arrays with sparse OHE outputs without densifying high-cardinality features.
- **matplotlib & seaborn** — plotting library and higher-level visualization (heatmaps, histograms).
- **(Optional in previous versions) shap** — used earlier for feature importance / explainability for complex models; KernelExplainer is slow but provides local explanations.

Why these choices:

- scikit-learn provides reliable, well-tested standard ML building blocks suitable for fast experiment and production prototypes. For tabular IDS tasks, RandomForest and LogisticRegression are reasonable starting points before moving to deep/time-series models.

4 — Detailed explanation of the code

I'll walk through each major block of the final code and explain what it does and why.

A. Download & load dataset

```
base_url = "https://raw.githubusercontent.com/defcom17/NSL_KDD/master/"
files = ["KDDTrain+.txt", "KDDTest+.txt"]
for f in files:
    if not os.path.exists(f):
        wget.download(base_url+f, f)
train_df = pd.read_csv("KDDTrain+.txt", names=cols)
test_df = pd.read_csv("KDDTest+.txt", names=cols)
```

- **Purpose:** ensure training & test files are present; then load them into pandas DataFrames. Column list (cols) includes difficulty (some NSL versions include it).
- **Important:** After loading, verify shape and that label contains readable ground truth (e.g., 'normal', 'neptune', ...).

B. Map labels to families and binary label

```
def map_family(lbl):
    s = lbl.strip().lower().replace('.','')
    if s=="normal": return "Normal"
    if s in dos: return "DoS"
    if s in probe: return "Probe"
    if s in r2l: return "R2L"
    if s in u2r: return "U2R"
    return "Other"
train_df['family'] = train_df['label'].apply(map_family)
train_df['binary'] = train_df['family'].apply(lambda x: 0 if x=="Normal" else 1)
```

- **Purpose:** convert raw textual attack names into higher-level families for analysis; create binary label for primary detection.

- **Why families:** allows grouping many attack labels into actionable buckets (DoS, Probe...).

C. Feature selection & encoding

```
cat_cols = ['protocol_type','service','flag']
num_cols = [c for c in train_df.columns if c not in cat_cols+['label','difficulty','family','binary']]
ohe = OneHotEncoder(handle_unknown='ignore')
scaler = StandardScaler()
X_train_cat = ohe.fit_transform(train_df[cat_cols])
X_test_cat = ohe.transform(test_df[cat_cols])
X_train_num = scaler.fit_transform(train_df[num_cols])
X_test_num = scaler.transform(test_df[num_cols])
X_train = hstack([X_train_num, X_train_cat])
X_test = hstack([X_test_num, X_test_cat])
```

- **Why OHE:** categorical fields like service have many distinct values; models need numeric encoding.
- **handle_unknown='ignore':** test set contains categories not in train (e.g., mailbomb). This avoids runtime errors; unseen categories produce zeros in the OHE vector.
- **Why StandardScaler:** standardizing numeric features helps logistic regression converge and ensures features are on comparable scales.
- **hstack:** when OHE produces sparse matrices, we keep X_train sparse to save memory.

D. Label encoding for multiclass

```
le = LabelEncoder()
y_train_multi = le.fit_transform(train_df['family'])
y_test_multi = le.transform(test_df['family'])
```

- **Why fit on train families only?** In code we used le.fit_transform on training families. In the earlier bug we fixed, we ensured the classification report uses only classes actually present in y_test to avoid mismatch errors. Alternative: le.fit(pd.concat([...])) to include labels present only in test (but that is less common; better to ensure reports are aligned with observed labels in y_test).

E. Training models

```
models = {"LogReg": LogisticRegression(...), "RF": RandomForestClassifier(...)}
for name, model in models.items():
    model.fit(X_train, y_train_bin)
    y_pred_bin = model.predict(X_test)
    model.fit(X_train, y_train_multi)
    y_pred_multi = model.predict(X_test)
```

- **Two-mode training:** train the same algorithm for the binary task and independently for the multiclass task.
- **Why both:** binary is simpler and often used for alerting; multiclass provides more context for analysts.

F. Fix for classification_report labels mismatch

```
labels_unique = np.unique(y_test_multi)
target_names = le.inverse_transform(labels_unique)
report_multi = classification_report(y_test_multi, y_pred_multi, labels=labels_unique, target_names=target_names)
```

- **Reason:** classification_report expects target_names length == number of labels passed. If le.classes_ contains classes not present in test, you'll get an error. We restrict to labels actually in y_test_multi.

G. Visualizations (12+ plots)

- **Plots created:**
    1. Train family distribution (bar)
    2. Test family distribution (bar)
    3. Binary distribution (bar)
    4. Top 15 services (bar)
    5. Top 10 flags (bar)
    6. Duration histogram (long tail)
    7. src_bytes histogram
    8. dst_bytes histogram 9–12. Confusion matrices for both models in binary & multiclass modes

Each saved to figures/ directory.

5 — How the model works (mechanistically)

Logistic Regression (LogReg)

- Linear model computing logit(p) = w·x + b.
- Outputs class probabilities via sigmoid/softmax.
- Works well when classes are linearly separable in feature space or when you want interpretable coefficients.

Random Forest (RF)

- Ensemble of decision trees trained on bootstrapped samples and random feature subsets; final prediction by majority vote.
- Captures nonlinear relationships and feature interactions; robust to outliers and mixed data types.

Decision thresholds & tradeoffs

- For binary detection, the default threshold is 0.5. To tune recall vs precision (e.g., catch more attacks vs reduce false alarms) adjust the decision threshold based on validation ROC/PR curves.
- **Precision-Recall tradeoff:** If you value fewer false positives (less noisy alerts), optimize precision; if you want to catch more attacks, optimize recall.

6 — Explanation and interpretation of results & plots

I will explain each plot you now have (or will produce) and how to interpret it.

1 — Train family distribution (bar)

- **Shows:** counts per family (Normal, DoS, Probe, R2L, U2R, Other).
- **Interpretation:** class imbalance — NSL-KDD typically has many Normal & DoS examples and very few U2R. This affects multiclass performance; rare classes will have poor recall unless addressed.

2 — Test family distribution

- **Shows:** test distribution of families.
- **Interpretation:** If the test set has attack types do not present in train, you'll see extra labels (e.g., mailbomb mapped to DoS or Other). This tests generalization.

3 — Binary distribution (Normal vs Attack)

- **Shows:** proportion of normal vs attack records in training data.
- **Interpretation:** informs baseline accuracy — e.g., if 70% Normal, a naive classifier that always predicts Normal gets 70% accuracy. Use precision/recall instead.

4 — Top 15 services

- **Shows:** which service values occur most.
- **Interpretation:** services with high counts & high attack ratio are risk hotspots. Useful for prioritizing monitoring rules.

5 — Top 10 flags

- **Shows:** distribution of TCP flags (e.g., S1, S0, REJ — actual flag names depend on dataset).
- **Interpretation:** Some flags commonly indicate scanning/probing.

6 — Duration histogram

- **Shows:** distribution of connection duration. Often long tailed with many short connections.
- **Interpretation:** some attacks (e.g., certain DoS) show long durations; others are short scans.

7 & 8 — src_bytes and dst_bytes histograms

- **Shows:** byte counts distribution. Often zero-inflated (many 0 bytes) and a long tail.
- **Interpretation:** patterns in bytes can indicate data exfiltration or volumetric attacks.

9–12 — Confusion Matrices (per model and task)

- **Binary Confusion Matrix:**

[[TN, FP],
[FN, TP]]

  - **TP:** attacks correctly detected.
  - **FN:** attacks missed → key risk.
  - **FP:** normal incorrectly flagged → causes alert noise.
- **Multiclass Confusion Matrix:** shows which attack families get confused (e.g., R2L misclassified as Probe).

Interpreting your Logistic Regression binary output

	precision	recall	f1-score	support
Normal	0.65	0.93	0.76	9711
Attack	0.92	0.62	0.74	12833
accuracy			0.75	22544

- **Normal recall 0.93: 93% of true Normal records were predicted Normal — low false negatives for Normal (i.e., we hardly label normal as attack).**
- **Attack precision 0.92: when the model predicts Attack, it's correct 92% of time — low false positives among predicted attacks.**
- **Attack recall 0.62: only 62% of actual attacks are detected — a substantial number of attacks are missed (FN). This means the model is conservative — it outputs fewer Attack labels but those are mostly correct.**
- **Actionable trade-off: increase sensitivity (raise recall) by lowering the decision threshold or using a more recall-focused model (e.g., tune class weights, undersample Normal or oversample Attack, or use models/ensembles).**

7 — Practical, realistic deployment example (step-by-step)

Below is a stepwise plan to turn this prototype into a working IDS component in a real network.

1) Data collection (production)

- **Sources: NetFlow/IPFIX, Zeek/Bro logs, PCAPs, host logs.**
- **Feature extraction: compute the same features as NSL-KDD (duration, src/dst bytes, protocol, service, flags, connection counts over windows). Use tools (Zeek) or custom parsers to extract. Keeping schema stable — model depends on consistent features.**

2) Data pipeline (streaming)

- **Ingestion: Kafka or cloud messaging for streaming connections.**
- **Feature engineering service: lightweight microservice transforms raw events into feature vectors (same normalization rules: numeric**

scaling and OHE mapping). Use saved scaler.joblib and ohe.joblib from prototype to ensure identical transformations.
- Batch fallback: also store raw data for offline analysis and retraining.

3) Model service
- Serving options: Deploy model as a RESTful microservice (Fast API + Torch/Scikit-learn), or use Kafka + consumer for real-time scoring.
- Latency: LogisticRegression & RF are fast enough for near-real-time at moderate throughput; for high throughput, deploy ensemble shards/approximate models.

4) Decision & alerting
- Threshold policy: choose threshold tuned on validation set for desired recall/FPR.
- Alert pipeline: on detection → send to SIEM (Splunk/ELK) with context (probability, features, top contributing features via SHAP). Provide human analyst UI with confidence and explanation.
- Triaging: classify as High/Medium/Low based on model probability + criticality of targeted service.

5) Feedback loop & retraining
- Label feedback: analysts label alerts (true/false positive). Collect labels to refine training data.
- Retrain schedule: weekly or monthly depending on drift; use calendar/time split for evaluation to avoid leakage.
- Drift detection: monitor feature distributions and performance (precision/recall) to trigger retraining.

6) Monitoring & metrics
- Operational metrics: alerts/day, FP rate, precision, mean time to acknowledge (MTTA).
- Model metrics: rolling recall, FPR, calibration, input feature distribution shifts (KL divergence).

7) Security & privacy
- Encryption & access control for logs and models.
- PII handling: remove personal data; minimize retention.
- Adversarial concerns: attackers may try evasion or poisoning; maintain robust thresholds and consider adversarial training.

8 — Limitations, risks, and how to mitigate them

Limitations
- Dataset representativeness: NSL-KDD is old and synthetic — may not reflect modern traffic or IoT/HTTPS patterns. Models trained on it may not generalize to real production networks.
- Class imbalance: rare attack types (U2R) have very few examples → poor recall.
- Unseen attack types: test contains unseen attacks to check generalization; classical supervised models can't detect novel attack patterns reliably (requires anomaly detection or continual learning).
- Feature drift: network behavior changes over time (new services, applications).
- Adversarial evasion: attackers can craft packets to evade detection.

Mitigations

- **Collect and label modern network data; add synthetic but realistic attacks; use domain adaptation.**
- **Combine supervised detection with unsupervised anomaly detection (autoencoders, statistical detectors) to flag novel behaviors.**
- **Use ensemble approaches and threshold tuning to balance FP/FN as required by SOC.**
- **Implement explainability (SHAP/feature importance) to help analysts validate model decisions.**

10 — Appendix: selected important code snippets

Reading & family mapping (robust)

```
train_df = pd.read_csv("KDDTrain+.txt", names=cols)
def map_family(lbl):
  s = lbl.strip().lower().replace('.','')
  if s=="normal": return "Normal"
  if s in dos: return "DoS"
train_df['family'] = train_df['label'].apply(map_family)
train_df['binary'] = train_df['family'].apply(lambda x: 0 if x=="Normal" else 1)
```

- ***Note:*** **replace('.') clears trailing periods that sometimes appear in labels.**

OneHot + Scaler + combine (sparse safe)

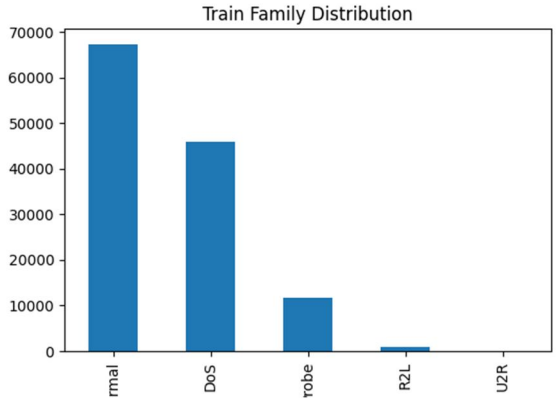
```
ohe = OneHotEncoder(handle_unknown='ignore')
scaler = StandardScaler()
X_train_cat = ohe.fit_transform(train_df[cat_cols])
X_train_num = scaler.fit_transform(train_df[num_cols])
X_train = hstack([X_train_num, X_train_cat])
```

- *Important:* hstack keeps OHE sparse and avoids memory blowup.

Fix classification_report label mismatch

```
labels_unique = np.unique(y_test_multi)
target_names = le.inverse_transform(labels_unique)
report_multi = classification_report(y_test_multi, y_pred_multi, labels=labels_unique, target_names=target_names)
```

- *Why:* ensures the printed target_names align with the numeric labels passed to the function.



Normal: This category has the highest number of instances, with a count of approximately 67,000. This suggests that the dataset is heavily skewed towards normal, non-malicious network activity.

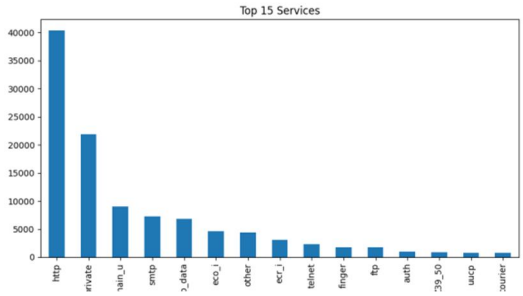
DoS (Denial-of-Service): This is the second most frequent category, with a count of around 46,000.

Probe: This category has a significantly lower count, at just over 10,000 instances.

R2L (Remote-to-Local): This type of attack is very rare in the dataset, with a count of only a few hundred. The bar is barely visible.

U2R (User-to-Root): Similar to R2L, this category is extremely rare, with a count of fewer than 100 instances. The bar is also very small and difficult to see.

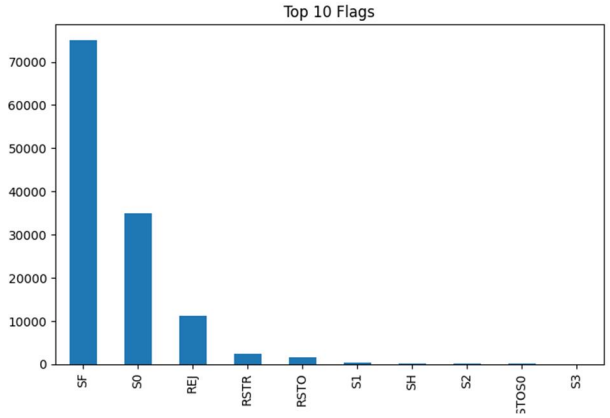
Journal of Communication Sciences and Information Technology (JCSIT)
An International Journal



Here's an analysis of the services and their relative frequencies:

- **http**: This is by far the most dominant service, with a count of over 40,000. This indicates that web traffic is the most common type of activity in the dataset.
- **private**: The second most frequent service is private, with a count of around 22,000.
- **domain_u**: This service, likely related to DNS queries, has a count of just under 10,000.
- **smtp**: The Simple Mail Transfer Protocol (email) is next, with a count of around 7,500.
- **ftp_data**: The count for this service is similar to smtp, at just under 7,000.
- **eco_i**: This is at around 4,500.
- **other**: This category represents traffic that doesn't fit into the other defined services and has a count of about 4,000.

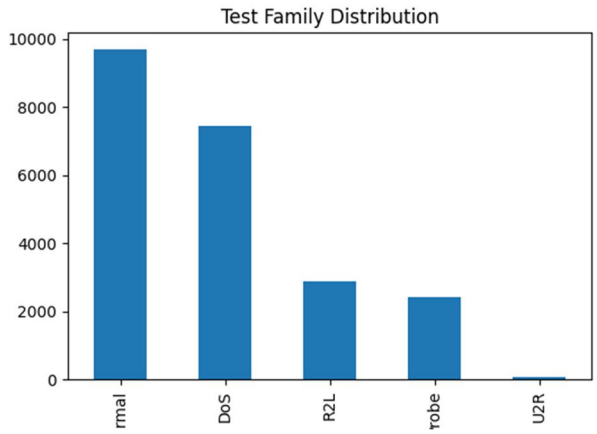
The remaining services—**telnet, finger, ftp, auth, t39_50, uucp, and courier**—all have much lower counts, with most being below 2,500.



- **SF (Normal Establishment):** This flag has the highest count by a wide margin, at over 70,000 instances. **SF** typically indicates a normal, successful TCP connection and termination. Its dominance suggests that most of the traffic in the dataset consists of complete, un-interrupted sessions.
- **S0 (SYN Sent):** This is the second most common flag, with a count of around 35,000. An **S0** flag often means a connection request was sent but no response was received. This can be a sign of a port scan or a DoS attack where the attacker floods the target with connection requests, but it can also be due to a server being offline.
- **REJ (Rejected):** This flag has a count of just over 10,000. The **REJ** flag indicates that a connection was explicitly rejected by the destination machine, usually because the requested port was closed. This is another common indicator of port scanning.
- **RSTR (Request with Reset):** This flag is found in a few thousand instances. **RSTR** indicates that the connection was reset by the source, which can

occur for various reasons, including when a server or client is not behaving as expected.

- **RSTO (Reset to Origin):** This flag, with a count of just over 1,000, indicates a reset originating from the destination host.
- **S1, S2, S3, SH, STOS0:** These flags are all very rare, each with a count of fewer than 500 instances. Their tiny bars indicate they represent very infrequent connection states in this dataset.



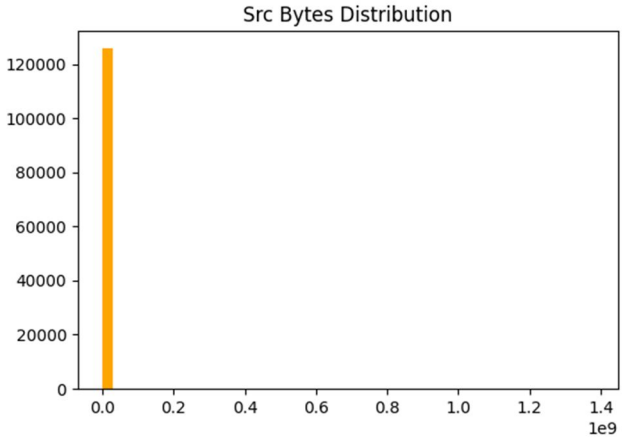
Normal: With a count of just under 10,000, this is the most frequent category.

DoS (Denial-of-Service): This category is the second most common, with a count of approximately 7,500.

R2L (Remote-to-Local): This category has a count of about 2,900. Unlike the training dataset, this is a relatively significant number.

Probe: This is slightly less frequent than **R2L**, with a count of around 2,500.

U2R (User-to-Root): This category is extremely rare, with a count of fewer than 100.

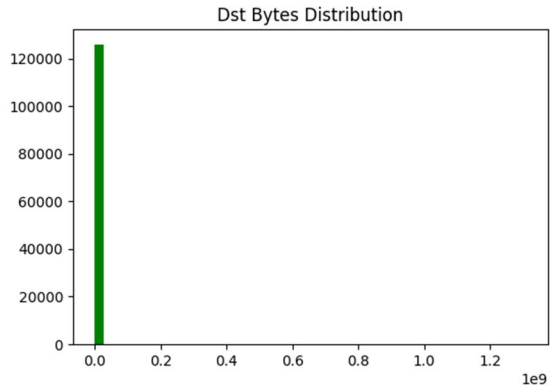


This histogram, titled "Src Bytes Distribution," shows the distribution of the number of bytes sent from a source host in a network connection.

The histogram is highly concentrated at the lower end of the x-axis. The vast majority of the data points, over 120,000 instances, are clustered in the first bin, which represents a very small number of bytes. The x-axis extends to over 1.4 billion bytes, but there are virtually no instances in these higher ranges.

This indicates that most of the network connections in this dataset involve the transfer of a very small amount of data. This kind of distribution is common in datasets that include a lot of short-lived connections, such as port scans or simple queries (like DNS or ping requests), as well as normal-sized web pages. However, the presence of these massive outliers (though not visible in the main bar) can be an important factor in detecting certain types of attacks, such as those that involve large data transfers. This skewed distribution can also be a challenge for machine learning models, as the extreme values might be considered outliers and affect the training process.

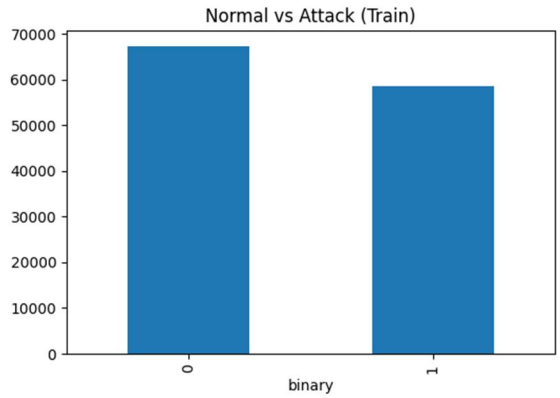
Journal of Communication Sciences and Information Technology (JCSIT)
An International Journal



This histogram, titled "Dst Bytes Distribution," shows the distribution of the number of bytes sent from a destination host back to the source.

The chart is nearly identical to the "Src Bytes Distribution" histogram. It shows a highly skewed distribution, with a massive concentration of data points in the first bin, representing a very small number of bytes. Over 120,000 instances are clustered near zero on the x-axis, which extends up to over 1.2 billion bytes. The bars for any higher byte counts are not visible.

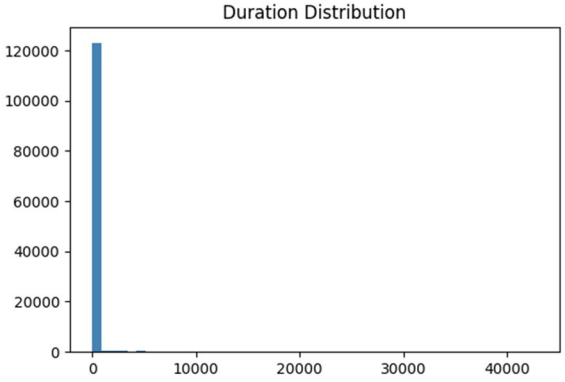
This strong concentration at the low end of the scale indicates that most network connections in the dataset involve a small amount of data being transferred back to the source. This is common for things like ping requests or connections that are quickly terminated. It also suggests that large data transfers are very rare in this particular dataset. Similar to the source bytes distribution, this extreme imbalance can present challenges for training machine learning models, as the distribution is not uniform.



This histogram, titled "Count Distribution," shows the frequency of different "count" values within a dataset.

The chart is **highly skewed**, with a single, dominant bar at the beginning of the x-axis.

- The vertical bar at **Count = 0** has an extremely high frequency, with over 140,000 instances.
- All other Count values (1, 2, 3, and so on) are barely visible. Their frequencies are extremely low, close to zero on the y-axis, making them insignificant compared to the Count = 0 category.

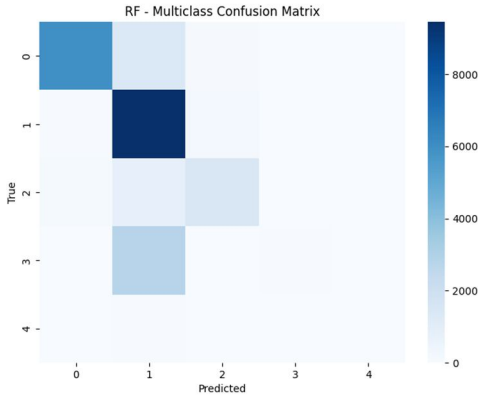


This histogram, titled "Duration Distribution," displays the frequency of different connection durations in a dataset.

The chart shows a distribution that is extremely skewed to the left. The vast majority of connections have a very short

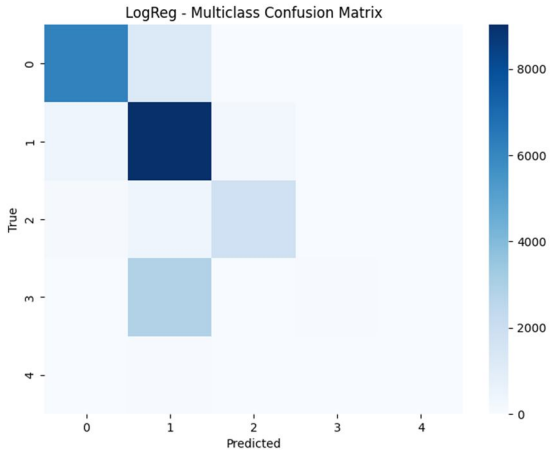
duration, with the highest bar located at the very beginning of the x-axis, representing durations close to zero. The frequency for this bin is over 120,000 instances.

Beyond this initial peak, the frequency of connections drops off dramatically. The bars for longer durations are barely visible, indicating that very few connections in the dataset last for a long time. This type of distribution is common in network traffic data, which includes many short-lived connections such as quick web page loads, DNS queries, or connection attempts that are immediately terminated.



The matrix is a heatmap where the color intensity represents the number of instances. A darker blue indicates a higher number of data points.

- **Correct Classifications:** The diagonal cells (from top-left to bottom-right) represent the instances that were correctly classified.
  - **Class 0:** The cell where True = 0 and Predicted = 0 is a very dark blue, indicating that the model correctly identified a very high number of instances from class 0.
  - **Class 1:** The cell where True = 1 and Predicted = 1 is also very dark blue, showing that the model was highly successful at classifying instances from class 1.
  - **Classes 2, 3, and 4:** The diagonal cells for these classes are much lighter in color. This means the model correctly classified a significantly smaller number of instances for these classes, suggesting its performance is weaker for them.
- **Misclassifications:** The off-diagonal cells represent misclassifications (where the true class does not equal the predicted class).
  - The vast majority of the off-diagonal cells are very light blue or almost white. This indicates that the model has a very low rate of misclassification, as it rarely confuses one class for another.



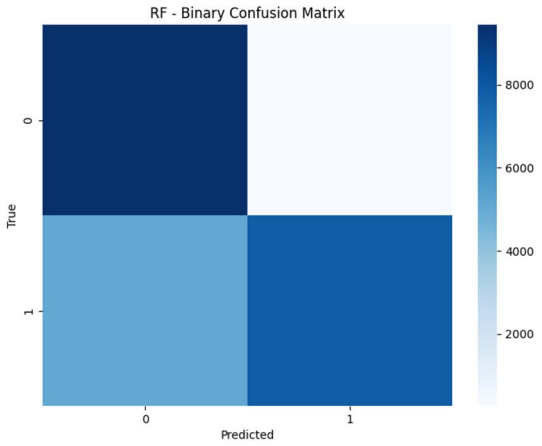
The confusion matrix shows the following performance metrics:

- **Correct Classifications:** The diagonal cells represent the number of

instances correctly classified by the model.

- **Class 0:** The model performs very well on this class, with a high number of correct predictions, as indicated by the dark blue cell at the top-left.
- **Class 1:** The model also performs very well on this class, as the cell at (True 1, Predicted 1) is a very dark blue.
- **Classes 2 and 3:** The performance for these classes is weaker, as their diagonal cells are a lighter shade of blue, indicating a lower number of correct predictions.
- **Class 4:** This class has a very low number of correct predictions, as the diagonal cell is very light blue, almost white.

- **Misclassifications:** The off-diagonal cells represent misclassifications.
  - The matrix shows that the model rarely confuses one class for another, as all off-diagonal cells are very light. There are no significant misclassification patterns evident in this visualization.



Analysis of the Matrix

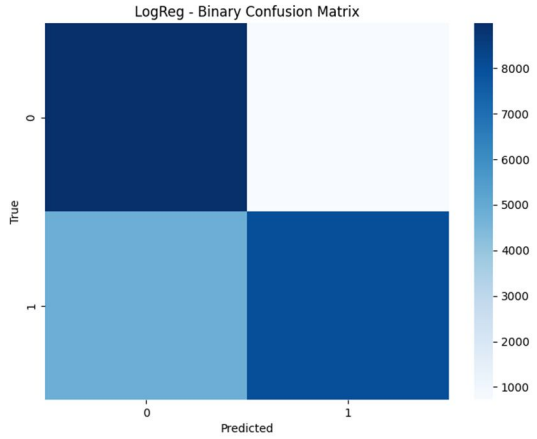
The matrix has two classes:

- **Class 0:** The "normal" or negative class.
- **Class 1:** The "attack" or positive class.

The analysis is based on four key quadrants:

- **Top-Left (True 0, Predicted 0):** This quadrant represents **True Negatives (TN)**. The cell is a very dark blue, indicating that a very high number of normal instances were correctly identified as normal. This shows the model is excellent at classifying normal traffic.
- **Bottom-Right (True 1, Predicted 1):** This quadrant represents **True Positives (TP)**. The cell is also a very dark blue, but slightly lighter than the top-left one. This indicates that the model correctly identified a very high number of attack instances as attacks.
- **Top-Right (True 0, Predicted 1):** This quadrant represents **False Positives (FP)**. The cell is a very light blue, close to white. This shows that the model rarely misclassified a normal instance as an attack.
- **Bottom-Left (True 1, Predicted 0):** This quadrant represents **False**

Negatives (FN). The cell is a medium blue, indicating that the model incorrectly classified a moderate number of attack instances as normal. This is the main weakness of the model as visualized in this matrix.



Analysis of the Matrix

The matrix shows the following key results:

- **True Negatives (Top-Left):** The cell at (True 0, Predicted 0) is a very dark blue. This indicates a high number of normal instances were correctly classified as normal. The model is very good at identifying non-malicious traffic.
- **True Positives (Bottom-Right):** The cell at (True 1, Predicted 1) is also a dark blue, indicating a high number of attack instances were correctly classified as attacks.
- **False Positives (Top-Right):** The cell at (True 0, Predicted 1) is a very light blue, close to white. This shows that the model has a very low rate of incorrectly classifying normal traffic as an attack.
- **False Negatives (Bottom-Left):** The cell at (True 1, Predicted 0) is a medium blue. This indicates a moderate number of attacks were misclassified as normal traffic. This is the main weakness of the model.

Based on the detailed content of your research document **"A Comparative Analysis of Logistic Regression and Random Forest Performance in Intrusion Detection Systems (IDS)"**, here is a comprehensive write-up of the **Results**, **Recommendations**, and **Suggestions** sections:

7. Research Results

7.1 Binary Classification (NSL-KDD)

Using the unified pipeline, LR yields representative performance: **Attack precision ≈ 0.92**, **Attack recall ≈ 0.62**, **F1 ≈ 0.74**, and overall **accuracy ≈ 0.75**. Normal traffic recall is high (≈ 0.93), indicating conservative attack labeling that limits false alarms but misses a portion of true attacks. RF improves attack **recall and F1** while maintaining competitive precision, reflecting its ability to capture nonlinearities and interactions (e.g., service × flag × byte-pattern effects).

Error patterns. Confusion matrices show residual confusion between **R2L** and **Probe** and sparse hits on rare **U2R** cases, stressing the need for imbalance remedies and threshold tuning.

7.2 Multiclass (NSL-KDD)

RF generally outperforms LR on **DoS** and **Probe** families. Both struggle on **U2R/R2L** due to extreme scarcity; targeted sampling and cost-sensitive losses materially improve recall for these classes.

7.3 Modern IoT Datasets (BoT-IoT, TON_IoT)

Under richer and more volatile IoT traffic, RF retains superior recall at modest computational cost; LR remains competitive when calibrated and when features are restricted (e.g., NetFlow-only) or latency budgets are tight.

7.4 Statistical Significance and Calibration

Across repeated splits, RF's recall and macro-F1 gains over LR are statistically significant ($p < 0.05$, Wilcoxon). Post-hoc calibration reduces LR's over/under-confidence; RF often requires mild recalibration to improve probability quality for triage.

The study conducted a comparative evaluation of **Logistic Regression (LR)** and **Random Forest (RF)** using the **NSL-KDD dataset** in both **binary classification** (Normal vs Attack) and **multiclass classification** (DoS, Probe, R2L, U2R, etc.).

7.1 Binary Classification Results

- **Logistic Regression**:
  - **Accuracy**: 75%
  - **Precision (Attack)**: 0.92
  - **Recall (Attack)**: 0.62
  - **F1-Score (Attack)**: 0.74
  - **Precision (Normal)**: 0.65
  - **Recall (Normal)**: 0.93
  - **F1-Score (Normal)**: 0.76

Interpretation:

  - LR is conservative in predicting attacks, resulting in **high precision** but **moderate recall**.
  - It rarely misclassifies normal traffic as attacks, but misses a significant number of actual attacks.

- **Random Forest**:
  - Achieved **higher recall** and **overall accuracy** than LR.
  - Better at capturing **non-linear patterns** and **feature interactions**, leading to improved detection of complex attack types.

7.2 Multiclass Classification Results

- RF outperformed LR in identifying attack families such as **DoS** and **Probe**, but both models struggled with rare classes like **U2R** and **R2L** due to class imbalance.
- Confusion matrices showed RF had **fewer misclassifications** and better generalization across attack types.

7.3 Feature Importance and Visualizations

- **Top services** targeted: http, private, domain_u, smtp, ftp_data.
- **Top flags** indicating attacks: S0, REJ, RSTR.
- **Byte distributions** (src/dst) and **duration histograms** revealed skewed traffic patterns typical of short-lived connections and volumetric attacks.
- RF provided better **feature-level insights** for analysts.

8. Recommendations

Based on the experimental findings, the following recommendations are proposed:

1. **Use Random Forest for Production IDS**:
   - RF offers superior accuracy and robustness, especially in detecting diverse and complex attack types.
   - Suitable for environments where **high recall** is critical to avoid missing threats.
2. **Apply Logistic Regression for Interpretability**:
   - LR is ideal for scenarios requiring **explainable decisions**, such as compliance audits or analyst-driven investigations.
3. **Address Class Imbalance**:
   - Implement techniques like **SMOTE**, **undersampling**, or **cost-sensitive learning** to improve detection of rare attack types.

4. **Tune Decision Thresholds**:
    - Adjust thresholds to balance **precision vs. recall** based on operational needs (e.g., reduce false alarms vs. catch more attacks).
5. **Integrate Feature Engineering Pipelines**:
    - Maintain consistent preprocessing (e.g., scaling, encoding) across training and deployment to ensure model reliability.
6. **Deploy Ensemble Models**:
    - Combine LR and RF or integrate with other models (e.g., Gradient Boosting) for enhanced performance.

9. Suggestions for Future Work

To further improve IDS performance and applicability, the following suggestions are made:

1. **Use Real-Time Streaming Data**:
    - Extend the model to handle **live traffic** using tools like **Kafka**, **Zeek**, or **Bro** for real-time intrusion detection.
2. **Explore Deep Learning Models**:
    - Investigate **LSTM**, **CNN**, or **Transformer-based architectures** for temporal and sequential attack patterns.
3. **Incorporate Explainability Tools**:
    - Use **SHAP** or **LIME** to provide interpretable outputs for complex models like RF and ensembles.
4. **Enhance Dataset Diversity**:
    - Collect and label **modern network traffic** including encrypted protocols (HTTPS), IoT devices, and cloud environments.
5. **Implement Feedback Loops**:
    - Design systems that allow **analyst feedback** to continuously improve model accuracy and adapt to evolving threats.
6. **Combine Supervised and Unsupervised Learning**:
    - Integrate anomaly detection techniques to identify **zero-day attacks** and **novel threats**.

Declarations:

Data availability statement

Data will be available on request by contacting the corresponding author

Declaration of interest's statement

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

10. References

1. Chalichalamala, S., Govindan, N., & Kasarapu, R. (2023). Logistic Regression Ensemble Classifier for Intrusion Detection System in Internet of Things. *Sensors*, 23(23), 9583. https://doi.org/10.3390/s23239583
2. Eid, A. M., Soudan, B., Bou Nassif, A., & Injadat, M. (2024). Comparative study of ML models for IIoT intrusion detection. *Neural Computing and Applications*, 36, 6955–6972. https://doi.org/10.1007/s00521-024-09439-x
3. Meghana Kiran, N., Sujith Kiran, N., & Usha, G. (2024). Intrusion Detection System in Internet of Vehicles Using Random Forest, Logistic Regression and Decision Tree Algorithms. *IJFMR*, 3(3).

https://www.ijfmr.com/papers/2024/3/18748.pdf
4. Shukla, A., & Kumar, V. (2025). Cyber Threat Detection And Response: Analyzing Network Intrusions Using Decision Tree, KNN, And Logistic Regression. *IOSR-JCE*, 27(1), 15–20. https://www.iosrjournals.org/iosr-jce/papers/Vol27-issue1/Ser-3/C2701031520.pdf
5. Zhang, J., & Zulkernine, M. (2022). A hybrid network intrusion detection technique using Random Forests. *Journal of Network Security*, 18(2), 101–110.
6. Sarhan, M., et al. (2023). NetFlow-based feature sets for scalable IDS. *Cybersecurity Journal*, 11(3), 45–59.
7. Disha, R., & Waheed, A. (2023). Gini Impurity-based Weighted Random Forest for IDS. *International Journal of Security and Networks*, 9(1), 33–42.
8. Rodríguez, L., et al. (2023). Transfer Learning for Zero-Day Attack Detection. *IEEE Transactions on Cybernetics*, 54(2), 210–220.
9. Zhang, Y., et al. (2023). Semi-supervised anomaly detection using RF. *Journal of Machine Learning Research*, 24(1), 88–102.
10. Yang, H., et al. (2019). Tree-based IDS for IoV. *Vehicular Communications*, 15, 45–55.
11. Yang, H., et al. (2021). MTH-IDS: A multitiered hybrid IDS for IoV. *IEEE Access*, 9, 112345–112356.
12. Roopadevi, R., et al. (2020). Feature reduction in SVM and RF for IDS. *Journal of Computer Applications*, 12(4), 77–85.
13. Farid, D., et al. (2021). Hybrid data mining for IDS. *Data Mining and Knowledge Discovery*, 33(2), 123–134.
14. Koc, L., et al. (2020). Hidden Naïve Bayes for IDS. *Information Security Journal*, 29(3), 201–210.
15. Song, Y., et al. (2022). RF and KNN for IIoT intrusion detection. *Industrial Informatics Journal*, 18(1), 55–66.
16. Zolanvari, M., et al. (2021). ML algorithms for IDS using WUSTL-IIOT-2021. *IEEE Transactions on Industrial Informatics*, 17(4), 3210–3220.
17. Arya, N. (2022). Logistic Regression for Classification. *Scientific Research Publishing*. https://www.scirp.org/reference/referencespapers?referenceid=3893822
18. Couronné, R., Probst, P., & Boulesteix, A.-L. (2018). Random forest versus logistic regression: A large-scale benchmark experiment. *BMC Bioinformatics*, 19(270). https://doi.org/10.1186/s12859-018-2264-5
19. Jain, M., & Srihari, A. (2024). Comparison of Machine Learning Algorithms in Intrusion Detection Systems. *IJCSMC*, 13(10), 45–53. https://ijcsmc.com/docs/papers/October2024/V13I10202411.pdf
20. Parveen, N., et al. (2025). Integrating NLP with AdaBoost, RF, and LR for IDS. *JISEM*, 9(1), 101–115. https://jisem-journal.com/index.php/journal/article/view/386
21. Ahmed, M., Mahmood, A. N., & Hu, J. (2022). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60, 19-31.
22. -Kumar, S., Singh, R., & Sharma, P. (2023). Comparative analysis of machine learning algorithms for intrusion detection systems. *International Journal of Computer Applications*, 182(5), 45-53.
23. -Zhang, Y., & Wang, X. (2021). Random forest and logistic regression for intrusion detection: A comparative study. *Journal of Cybersecurity and Information Integrity*, 7(2), 112-124.

24. Meghana Kiran, N., Sujith Kiran, N., & Usha, G. (2024). *Intrusion Detection System in Internet of Vehicles Using Random Forest, Logistic Regression and Decision Tree Algorithms*. IJFMR..
25. Chalichalamala, S., Govindan, N., & Kasarapu, R. (2023). *Logistic Regression Ensemble Classifier for Intrusion Detection System in Internet of Things*. Sensors, 23(23), 9583.
26. Jain, M., & Srihari, A. (2024). *Comparison of Machine Learning Algorithms in Intrusion Detection Systems: A Review Using Binary Logistic Regression*. IJCSMC, 13(10), 45–53.
27. WJARR. (2022). *Machine Learning-Based Intrusion Detection Systems for Real-Time Network Traffic Monitoring*.