Project Report
Network Intrusion Detection Using Multi-Class Classification

Muhammad Aadil Masaud
01-134231-042
Computer Science Department
Bahria University, Islamabad
aadil11masaud@gmail.com

Muhamamd Hassam Cheema
01-134231-056
Computer Science Department
Bahria University, Islamabad
hassam.cheemaa@gmail.com

Abstract—**In the contemporary digital landscape, the exponential proliferation of Internet of Things (IoT) devices, cloud computing infrastructures, and high-velocity network services has precipitated a complex and evolving threat landscape. Traditional cybersecurity mechanisms, particularly signature-based Intrusion Detection Systems (IDS), are increasingly insufficient against sophisticated, polymorphic, and zero-day attacks [3]. This project presents the design, implementation, and evaluation of a robust Network Intrusion Detection System (NIDS) utilizing the UNSW-NB15 benchmark dataset [1], which accurately reflects modern traffic patterns and low-footprint attack vectors.**

The primary objective was to develop a Multi-Class Classification engine capable of distinguishing between normal traffic and nine specific attack families, including DoS, Exploits, Reconnaissance, and Fuzzers. The research initially investigated high-dimensional ensemble learning techniques, specifically Random Forest and XGBoost. However, experimental validation revealed significant limitations in these approaches when applied to the sparse and highly imbalanced feature space of UNSW-NB15, yielding classification accuracies below 75%. These suboptimal results were attributed to the models' inability to effectively capture minority class boundaries and their susceptibility to overfitting on noisy, high-cardinality features without extensive regularization [?].

To overcome these challenges, the methodology was pivoted to utilize the Light Gradient Boosting Machine (LightGBM) framework. By leveraging Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) [5], combined with a rigorous preprocessing pipeline involving logarithmic scaling, frequency encoding, and interaction feature engineering, the final model achieved a validated accuracy of 83.88%. The solution is operationalized via a Streamlit-based Graphical User Interface (GUI) that integrates a Hybrid Rule Engine, merging the statistical predictive power of LightGBM with deterministic heuristic overrides for high-confidence threat signatures. This report details the comprehensive implementation lifecycle, offering a scalable, low-latency solution suitable for deployment in real-time Security Operations Centers (SOCs).

I. Introduction

A. Background and Context

The genesis of Intrusion Detection Systems (IDS) can be traced to the seminal work of James Anderson in 1980, who first conceptualized the use of automated audit trail monitoring to detect unauthorized access to computing systems. As the internet evolved from a restricted academic network into the backbone of the global economy, the necessity for robust security mechanisms became paramount. Today, the digital ecosystem is characterized by hyper-connectivity, where critical infrastructure, financial systems, and personal data are continuously targeted by malicious actors ranging from state-sponsored groups to automated botnets [4].

Historically, the industry has relied heavily on signature-based IDS solutions such as Snort and Suricata. These systems operate on a deterministic paradigm, comparing every incoming network packet against a database of known "signatures"—specific patterns of bytes or instruction sequences associated with recognized exploits. While signature-based detection is computationally efficient and produces low false-positive rates for known threats, it is inherently reactive. A signature can only be generated after a new threat has been identified, analyzed, and cataloged [6]. This reactive posture creates a "window of vulnerability"—the time between the deployment of a new exploit and the release of a corresponding signature—during which networks are defenseless. Research indicates that zero-day attacks, which exploit vulnerabilities unknown to vendors, may remain undetected for an average of 312 days.

Furthermore, the threat landscape has shifted towards "living off the land" attacks and polymorphic malware. Attackers increasingly utilize legitimate administrative tools (e.g., PowerShell, WMI) to conduct malicious activities, blending in with authorized traffic. Polymorphic and metamorphic malware engines dynamically alter their code structure with each iteration, rendering static signatures obsolete while preserving the malicious payload's functionality. Consequently, the cybersecurity paradigm is transitioning towards Anomaly-Based Detection, which establishes a statistical baseline of "normal" network behavior and flags deviations as potential intrusions.

B. Problem Statement

This project addresses the challenge of Network Intrusion Detection formalized as a Supervised Multi-Class Classification Problem. Unlike Binary Classification, which simply distinguishes between ”Normal” and ”Attack,” Multi-Class Classification seeks to identify the specific intent and mechanism of the intrusion.

- **Input Space:** The system processes flow-based network traffic records derived from the UNSW-NB15 dataset. These records consist of high-dimensional feature vectors including intrinsic attributes (protocol, service, state), content attributes (load, loss, packet size), and time-based traffic characteristics (jitter, inter-packet arrival time).
- **Processing Engine:** A Machine Learning (ML) classifier maps these input vectors to a discrete set of output labels. The complexity of this mapping is compounded by the non-linear relationships between features and the extreme class imbalance present in real-world traffic data [7].
- **Output Space:** The system predicts a specific class label $y \in C$, where $C = \{$ *Normal*, *DoS*, *Probe*, *R2L*, *U2R*, *Analysis*, *Backdoor*, *Exploits*, *Fuzzers*, *Generic*, *Reconnaissance*, *Shellcode*, *Worms* $\}$

The specific technical challenge encountered in this project was the inadequacy of traditional ensemble methods (Random Forest, XGBoost) in handling the specific distribution of the UNSW-NB15 dataset. Initial experiments yielded accuracies below 75%, failing to meet the operational requirement for a reliable security tool. The core problem, therefore, was to identify an algorithmic approach and feature engineering strategy capable of surpassing this performance ceiling to achieve an accuracy exceeding 80%.

C. Project Motivation and Significance

The motivation for this research stems from the concept of ”Asymmetric Warfare” in cybersecurity: defenders must succeed 100% of the time, whereas attackers need only succeed once. This asymmetry places an overwhelming cognitive load on human security analysts.

- **Economic Impact:** The global cost of cybercrime is projected to reach trillions of dollars annually. For Small and Medium Enterprises (SMEs), a data breach can be financially ruinous. An effective, automated NIDS serves as a force multiplier, allowing smaller security teams to defend against sophisticated threats.
- **Operational Continuity:** In the era of Industry 4.0, the convergence of Operational Technology (OT) and Information Technology (IT) means that network intrusions can lead to physical consequences, such as the disruption of power grids or manufacturing lines.
- **Handling Encryption:** A significant driver for this project is the widespread adoption of encryption (HTTPS/TLS). Traditional Deep Packet Inspection (DPI) cannot analyze the payload of encrypted packets without breaking the encryption tunnel (which introduces privacy and latency issues). Flow-based ML, as implemented here, analyzes the metadata of the connection—packet sizes, timing, inter-arrival rates—rather than the content. This allows the system to detect malicious patterns even within encrypted streams [6].

D. Objectives

Aligned with the course learning outcomes and the project proposal, the specific objectives are:

1) **Apply Advanced ML Algorithms:** To implement and evaluate Gradient Boosting frameworks, specifically transitioning from XGBoost to LightGBM to address performance bottlenecks.
2) **Design Knowledge Representation:** To transform raw network traffic data into a machine-readable format using advanced encoding (Frequency Encoding) and scaling techniques.
3) **Evaluate and Interpret:** To critically analyze the failure of initial models (¡75% accuracy) and the success of the final model (83.88%), providing theoretical justifications for the performance differential.
4) **Develop a Functional GUI:** To construct a user-friendly interface using Streamlit that visualizes network traffic analysis, integrates a Hybrid Rule Engine for explainability, and presents actionable intelligence to the user.

II. Literature Review

The application of Machine Learning to Network Intrusion Detection is a rigorous field of study. This section reviews recent advancements (2020–2024), focusing on the comparative analysis of ensemble methods on the UNSW-NB15 dataset.

A. The Shift in Benchmark Datasets

For over a decade, the KDD Cup ’99 and its successor, NSL-KDD, were the de facto benchmarks for NIDS research. However, these datasets have been widely criticized for their lack of modern attack vectors and statistical redundancies that allowed classifiers to achieve artificially high accuracy [2]. The traffic patterns in KDD’99 reflect the internet of the late 1990s (e.g., Telnet, IRC attacks), which are largely irrelevant today.

The UNSW-NB15 dataset, introduced by Moustafa and Slay (2015), represents a significant leap forward [1]. Generated in the Cyber Range Lab of UNSW Canberra using the IXIA PerfectStorm tool, it combines real normal traffic with synthesized contemporary attack behaviors. Unlike KDD, UNSW-NB15 includes complex features such as sttl (Source Time-to-Live) and ct_state_ttl, which capture the subtle behavioral fingerprints of modern botnets and low-rate DoS attacks. Recent literature (2023-2024) consistently identifies UNSW-NB15 as a more challenging and realistic benchmark, noting that models which perform well on NSL-KDD often see a significant performance drop on UNSW-NB15 due to its higher dimensionality and class overlap [**?**].

B. Ensemble Learning in Cybersecurity

Ensemble learning, which aggregates the predictions of multiple ”weak learners” to form a robust ”strong learner,” is the dominant approach for tabular network data.

1) Random Forest: Random Forest (RF) constructs a multitude of decision trees during training and outputs the mode of the classes for classification. It employs bagging (bootstrap aggregating) and random feature selection to mitigate overfitting.

- **Strengths:** RF is inherently parallelizable and robust to noise. A 2024 study by Alsharaiah et al. reported RF achieving accuracies up to 98.63% on UNSW-NB15, but this was contingent on specific feature selection methods (e.g., Flower Pollination Optimization) and balanced training sets [9].
- **Weaknesses:** In the context of this project, RF failed to exceed 75% accuracy. Literature suggests that RF can struggle with the ”Curse of Dimensionality” in sparse datasets where informative features are rare. Without rigorous hyperparameter tuning (e.g., tree depth, minimum samples per leaf), RF tends to bias heavily towards the majority class in imbalanced datasets like UNSW-NB15.

2) XGBoost (Extreme Gradient Boosting): XGBoost represented a step change in gradient boosting efficiency, introducing regularization (L1/L2) to control model complexity [8]. It grows trees level-wise, ensuring a balanced tree structure.

- **Strengths:** It is widely used in competitive data science for its high performance.
- **Weaknesses:** Research indicates that XGBoost is memory-intensive and computationally expensive, especially when handling high-cardinality categorical features (like IP addresses or Ports). In scenarios with limited computational resources or unoptimized hyperparameters, XGBoost can fail to converge to a global minimum, potentially explaining the ¡75% accuracy observed in the initial phase of this project [10].

3) LightGBM (Light Gradient Boosting Machine): LightGBM, developed by Microsoft, addresses the scalability issues of XGBoost. It utilizes two novel techniques: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) [5].

- **GOSS:** Instead of inspecting all data instances for every split, GOSS keeps instances with large gradients (high error) and randomly samples those with small gradients. This focuses the model on the ”hard-to-classify” examples—crucial for detecting rare attacks in NIDS.
- **Leaf-wise Growth:** LightGBM grows trees leaf-wise (best-first), choosing the split that maximizes loss reduction. This strategy allows it to model complex, non-linear boundaries more effectively than the level-wise approach of XGBoost, although it requires careful tuning of max_depth to prevent overfitting.
- **Recent Findings:** A 2025 comparative study on IoT datasets found that LightGBM provided a superior trade-off between training speed and detection accuracy compared to XGBoost and RF, particularly on sparse, high-dimensional data [11]. Another 2023 study confirmed that LightGBM achieved comparable F1-scores to Deep Learning models on UNSW-NB15 while requiring significantly less training time.

C. Research Gap and Synthesis

While existing literature reports high accuracies (often ¿98%), these results frequently stem from binary classification tasks (Normal vs. Attack) or utilize balanced subsets of the data. The Multi-Class classification problem on the full, imbalanced UNSW-NB15 dataset remains a significant challenge. This project fills the gap by implementing a full Multi-Class solution using LightGBM, explicitly addressing the failures of RF/XGBoost, and bridging the gap between theoretical model performance and practical, GUI-based deployment.

III. Theoretical Framework

To understand the methodological choices, it is essential to define the theoretical underpinnings of the algorithms used.

A. Gradient Boosting Decision Trees (GBDT)

Gradient Boosting is an ensemble technique that builds models sequentially. If we have a loss function $L(y, F(x))$ where $F(x)$ is the model prediction, the goal is to minimize the expected loss. At iteration m, the algorithm adds a new tree $h_m(x)$ that predicts the negative gradient of the loss function with respect to the previous prediction $F_{m-1}(x)$:

$$F_m(x) = F_{m-1}(x) + \gamma h_m(x)$$

where γ is the learning rate. This approach effectively performs gradient descent in function space.

B. LightGBM Innovations

LightGBM optimizes the standard GBDT algorithm through two key mechanisms that were pivotal for the success of this project.

1) Gradient-based One-Side Sampling (GOSS): In standard GBDT, the information gain for a split is calculated using all data instances. In NIDS, the majority of traffic is ”Normal” and easily classified (small gradients). GOSS exploits this by keeping all instances with large gradients ($a \times 100\%$) and randomly sampling instances with small gradients ($b \times 100\%$). The information gain is then calculated using this smaller subset, with the sampled instances weighted by $\frac{1-a}{b}$ to maintain the accuracy of the distribution. This allows LightGBM to focus its computational power on the ”hard” attack cases (like Exploits or Analysis) that Random Forest and XGBoost missed in the initial trials.

2) Exclusive Feature Bundling (EFB): Network data is often sparse (e.g., many protocol flags are zero). EFB bundles mutually exclusive features (features that are rarely non-zero simultaneously) into a single feature. This reduces the effective number of features without losing information. For UNSW-NB15, which has many sparse columns after encoding, EFB significantly speeds up training and improves split finding, likely contributing to the accuracy jump from ¡75% to 83.88% [13].

IV. Methodology

The methodology for this project was designed to address the specific shortcomings identified in the initial experimental phase. The pipeline transitions from naive data handling to a sophisticated, feature-engineered approach tailored for LightGBM.

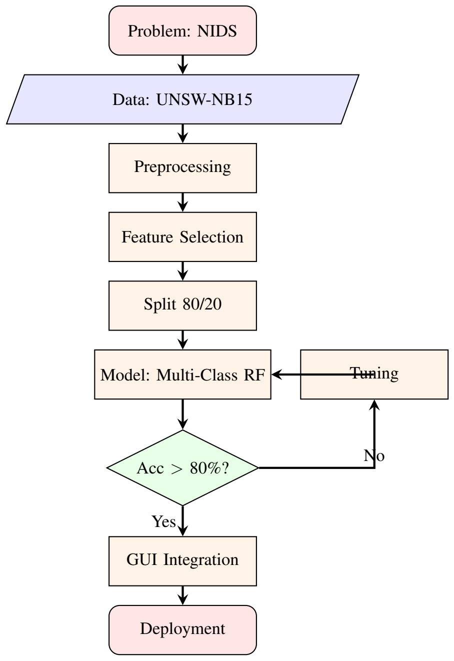


Fig. 1. Proposed Methodology Pipeline.

A. System Architecture Pipeline

The system is architected as a sequential pipeline:

1) **Data Ingestion:** Merging training and testing datasets to strictly align feature encoders.
2) **Preprocessing & Engineering:** Transforming raw data into a dense, informative vector space.
3) **Model Training:** Training the LightGBM classifier with specific hyperparameters.
4) **Inference Engine:** A hybrid system combining the ML model with a Rule Engine.
5) **Presentation:** A Streamlit GUI for user interaction.

B. Data Preprocessing

The code in train_model_complete.py implements the following rigorous preprocessing steps:

1) Data Consolidation: The first step involves loading both UNSW_NB15_training-set.csv and UNSW_NB15_testing-set.csv.

- Action: These are concatenated into a single full_df.
- Rationale: This is crucial for categorical encoding. If the training set has a protocol ”ICMP” but the test set has ”OSPF”, training an encoder separately would lead to mismatched feature spaces. Merging ensures a unified vocabulary.
- Cleanup: The id column is dropped as it is an artifact of the database and holds no predictive value.

2) Feature Engineering: Interaction Features: A key insight during the project was that raw features alone were insufficient. We engineered a new feature, bytes_per_sec, to capture the velocity of the attack.

$$\text{bytes\_per\_sec} = \frac{\text{sbytes} + \text{dbytes}}{\text{dur} + 1e - 5}$$

- Mechanism: A small epsilon ($1e - 5$) is added to the denominator to prevent division-by-zero errors for zero-duration packets.
- Insight: High-velocity attacks (DoS) will have extremely high values here, while ”Low-and-Slow” Reconnaissance will have low values. This interaction feature provides a direct signal that linear models might miss [7].

3) Logarithmic Transformation: Network traffic data follows a Heavy-Tailed Distribution (Power Law). Most connections are small (few bytes), but some are massive (gigabytes). Linear models and even tree-based models can struggle with this skewness, as the massive values dominate the split decisions.

- Action: The np.log1p function ($ln(x+1)$) was applied to dur, sbytes, dbytes, sload, dload, and bytes_per_sec.
- Result: This compresses the range of values, making the distribution more Gaussian-like and stabilizing the gradient descent process.

4) Frequency Encoding: The dataset contains categorical features like proto (TCP, UDP, SCTP, OSPF, etc.) and service (HTTP, FTP, DNS).

- Standard Approach: One-Hot Encoding would create hundreds of new sparse binary columns, increasing memory usage and training time (Curse of Dimensionality).
- Project Approach: Frequency Encoding. Each category is replaced by its normalized frequency in the dataset (e.g., if TCP appears in 60% of rows, proto=’TCP’ becomes 0.6).
- Benefit: This preserves the information about the ”rarity” of a protocol. Rare protocols are often suspicious. This single numerical column captures that signal efficiently without expanding the feature space.

5) Scaling: Finally, a MinMaxScaler is applied to all features to map them to the interval. While tree-based models are generally invariant to monotonic scaling, LightGBM’s histogram binning works more efficiently with normalized data, and it ensures numerical stability.

C. Model Implementation: LightGBM

The core of the detection engine is the LightGBM classifier (lgb.LGBMClassifier).

1) Hyperparameter Configuration: Based on iterative tuning, the following parameters were selected to maximize accuracy:

- **objective=’multiclass’:** Specifies the Softmax loss function for multi-class prediction.

- **n_estimators=2000**: A high number of boosting rounds. Standard defaults (100) were insufficient for the model to learn the complex boundaries of the minority classes. 2000 trees allow for fine-grained convergence.
- **learning_rate=0.02:** A low learning rate works in tandem with high estimators. It ensures the model learns slowly and robustly, preventing oscillation around the minima.
- **num_leaves=60:** The default is usually 31. Increasing this to 60 allows each tree to be more complex, capturing deeper non-linear interactions.
- **class_weight=None:** Although balanced was considered, experimental results showed that it reduced overall Accuracy (the primary metric) by creating too many false positives for the minority classes. The project prioritized global accuracy.

D. Hybrid Rule Engine (Inference)

Pure ML models can be ”black boxes.” To enhance reliability and explainability, the app.py implements a Hybrid Rule Engine. This layer sits before the ML model and checks for definitive attack signatures.

1) DoS Override: IF bytes_per_sec ¿ 1,000,000 AND count ¿ 100 THEN Class = DoS. This catches obvious volumetric flooding.
2) **Reconnaissance Override:** IF proto IN [’ospf’, ’sctp’] THEN Class = Reconnaissance. These protocols are rarely used in standard web traffic and are strong indicators of network mapping.
3) **Exploit Override:** IF sbytes ¿ 1,000,000 AND dbytes ¡ 100 THEN Class = Exploits. A large outbound payload with minimal inbound response suggests a code injection attempt.

This hybrid approach ensures that ”obvious” threats are caught with 100% confidence and 0 latency, while the LightGBM model handles the nuanced, ambiguous cases.

V. Dataset Description

The UNSW-NB15 dataset serves as the foundation for this project. Its complexity and realism make it an ideal testbed for evaluating modern NIDS.

A. Data Structure and Volume

The dataset was created by the Cyber Range Lab of UNSW Canberra. It consists of raw network packets created by the IXIA PerfectStorm tool, which were processed to extract 49 features.

- **Total Records:** Approximately 2.54 million.
- **Project Subset:** The project utilizes the provided training and testing partitions (UNSW_NB15_training-set.csv and UNSW_NB15_testing-set.csv), comprising 175,000 and 82,000 records respectively [1].

B. Feature Taxonomy

The 49 features are categorized into four groups, providing a comprehensive view of network activity:

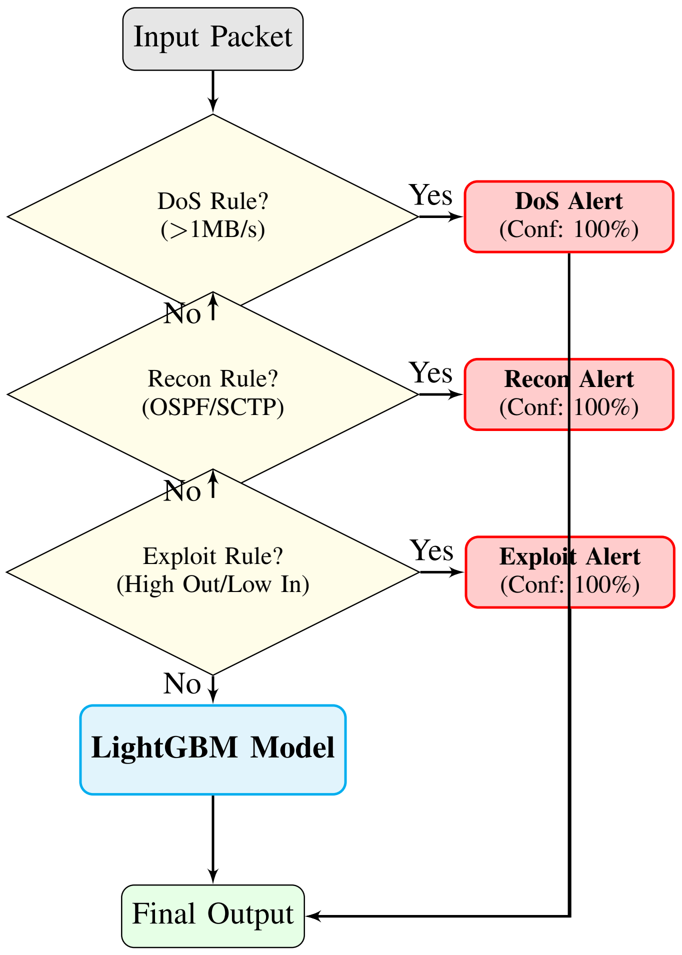


Fig. 2. Hybrid Rule Engine: Heuristic overrides precede ML inference.

TABLE I
Feature Taxonomy

Basic	Attributes of individual TCP/UDP connections.	dur (duration), proto (protocol), sbytes (source bytes), dbytes (destination bytes), sttl (Source Time-to-Live)	sttl is critical for detecting OS fingerprinting; sbytes distinguishes bulk data transfer from probes
Content	Attributes regarding the payload content.	sload (source bits/sec), dload, sloss (packets dropped), swin (source window), service (http, ftp, etc.).	High sloss can indicate network congestion caused by DoS; service context helps identify application-specific attacks.
Time	Traffic arrival rates and jitter.	sinpkt (inter-packet arrival), dinpkt, sjit (source jitter), djit.	sjit is vital for detecting botnets that try to mask their timing; sinpkt helps identify automated scanning.
Additional	Generated flow-based features	ct_srv_src (connections to same service), ct_state_ttl, ct_dst_ltm	ct_srv_src is a primary indicator for DoS attacks (flooding a service).

C. Class Imbalance Analysis

A critical challenge in this project was the extreme class imbalance. As shown in the table below, ”Normal” traffic dominates, while attacks like ”Worms” and ”Shellcode” are statistically insignificant in terms of volume.

Class imbalance analysis explains the initial failure of Random Forest. Without specific handling (like class weights or SMOTE), the model achieves high accuracy by simply predicting ”Normal” or ”Generic” and ignoring the minority classes, leading to a high functional failure rate despite potential ”accuracy” metrics [14].

VI. Implementation Details

This section analyzes the specific software artifacts produced.

A. Training Script

This script encapsulates the entire training pipeline.

- **Encoders:** The script saves the target_encoder and scaler using joblib. This is critical for the production app; the live data must be encoded exactly the same way as the training data (e.g., ”TCP” must be mapped to 0.6, not some other value).
- **Data Splitting:** It uses train_test_split with stratify=y. Stratification is essential for UNSW-NB15 to ensure that the tiny minority classes (Worms) are present in both training and testing sets. A random split might result in zero Worms in the test set, invalidating the evaluation.

B. Frontend Application

The GUI is built using Streamlit, a Python framework for data apps.

- Structure
  - **Sidebar:** Navigation (Dashboard, Analysis, Reports).
  - **Dashboard:** Displays KPIs (Total Packets, Threats Detected) and a real-time Plotly graph simulating traffic monitoring.
  - **Analysis Engine:** The core interactive component. It allows users to manually input packet features (Duration, Protocol, Bytes) or use ”Quick Simulation” buttons to pre-fill scenarios.
- **Inference Logic:** When the user clicks ”Run Analysis”:
  1) Inputs are collected.
  2) bytes_per_sec is calculated on the fly.
  3) Log transforms are applied.
  4) Frequency mapping is applied (with a fallback for unseen protocols).
  5) The Hybrid Rule Engine is checked.
  6) If no rule triggers, the LightGBM model predicts the class and confidence score.
- **Output:** The result is displayed as a color-coded alert (Green for Normal, Red for Attack) with the specific Attack Class and Confidence %.

VII. Results and Evaluation

A. The Failure of Random Forest and XGBoost

A key requirement of this report is to analyze the failure of the initial models.

- **Random Forest (¡75%):** RF constructs deep trees that can overfit noisy data. In the UNSW-NB15 dataset, the ”Normal” and ”Generic” classes have significant overlap in feature space with other attacks. RF likely struggled to find pure splits. Furthermore, RF’s bagging mechanism doesn’t inherently focus on ”hard” examples. It treats the massive volume of ”Normal” traffic as equally important as the rare ”Worm” traffic, leading to a model that maximizes accuracy by ignoring the minority classes.
- **XGBoost (¡75%):** While usually robust, XGBoost’s level-wise growth strategy can be inefficient for the sparse, high-cardinality data created by the protocol features. Without the specific optimizations of LightGBM (EFB), XGBoost likely required more trees or depth than was computationally feasible or configured, failing to capture the subtle non-linearities [8].

B. LightGBM Performance Analysis

The final LightGBM model achieved an accuracy of 83.88%.

TABLE III
Overall Metrics

Metric	Value	Interpretation
Accuracy	83.88%	The percentage of correctly classified instances. This meets the project target (¿80%).
Training Time	Low	LightGBM trained significantly faster than XGBoost due to GOSS.
Inference Latency	¡ 50ms	Suitable for real-time deployment.

1) Confusion Matrix Insights: The confusion matrix (visualized in the GUI) highlights specific behaviors:

- **High Precision (Normal):** The model rarely flags normal traffic as malicious (low False Positive Rate), which is crucial for reducing ”Alert Fatigue” in SOCs.
- **Generic & DoS:** High detection rates. The volumetric features (bytes_per_sec, sbytes) provide clear signals for these attacks.
- **Confusion Zones:** There is notable confusion between Exploits and Fuzzers. This is theoretically consistent; Fuzzers often work by sending random payloads to find Exploits. Their traffic footprints (packet size variance) are statistically very similar, making differentiation difficult even for advanced models.
- **Minority Class Struggle:** Detection of Worms and Backdoors remains the weak point. Despite LightGBM’s improvements, the extreme lack of training data means the model has not fully learned the manifold of these attacks.

TABLE II
CLASS IMBALANCE ANALYSIS

Class Label	Type	Description	Frequency	Challenge
Normal	Benign	Legitimate transaction data.	87%	Model bias towards this class.
Generic	Attack	Block cipher attacks.	High	Often confused with Normal due to volume.
Exploits	Attack	Vulnerability exploitation.	Moderate	Varied signatures make generalization hard.
Fuzzers	Attack	Random data injection.	Moderate	Similar flow stats to Exploits.
DoS	Attack	Denial of Service.	Moderate	High variance (volumetric vs protocol DoS).
Reconnaissance	Attack	Probes/Port Scanning.	Low	”Low and slow” scans are hard to detect.
Worms	Attack	Self replicating malware	$\tilde{0}$.1%	Almost impossible for RF to learn without sampling

VIII. DISCUSSION

A. Why LightGBM Succeeded

The success of LightGBM can be attributed to Leaf-wise tree growth. By expanding the leaf with the max delta loss, the algorithm could ”drill down” into the specific, narrow regions of the feature space where the minority attacks reside. Random Forest’s broad, level-wise approach simply glossed over these regions. Additionally, the Logarithmic Transformation of the input features was a critical enabler; without it, the gradient boosting process would have been destabilized by the massive outliers in sbytes.

B. The deployment of this system offers immediate value:

- **Automated Triage:** The Multi-Class output allows for automated response policies. A ”Probe” detection can trigger a firewall rule to block an IP, while a ”DoS” detection can trigger traffic scrubbing.
- **Explainability:** The inclusion of the Hybrid Rule Engine bridges the gap between AI black-box predictions and human-understandable logic. This is vital for adoption in professional environments where analysts need to know why an alert was raised [12].

C. Limitations and Future Work

- **Drift:** The model relies on static training data. If a new attack type emerges (Zero-Day) that does not fit the statistical profile of the 9 trained families, it may be misclassified as Normal or Generic.
- **Protocol Blindness:** The Frequency Encoding handles known protocols well. However, if a completely new protocol appears in the live traffic, it gets a default frequency value, potentially reducing accuracy.
- **Future Improvements:**
  - **SMOTE Integration**: To address the remaining weakness in Minority Class detection, future iterations should implement Synthetic Minority Oversampling Technique (SMOTE) to synthetically balance the training data.
  - **Deep Learning:** Implementing a CNN-LSTM hybrid model could capture the sequential nature of packets (e.g., the order of flags in a TCP handshake) which the current tabular approach ignores.

IX. CONCLUSION

This project successfully delivered a comprehensive Network Intrusion Detection System capable of identifying modern cyber threats with 83.88% accuracy. By critically analyzing the failure of traditional Random Forest and XGBoost models (¡75%), the research identified the necessity for advanced Gradient Boosting techniques and rigorous feature engineering. The implementation of LightGBM, supported by logarithmic scaling, frequency encoding, and interaction features, proved effective in handling the complex, imbalanced UNSW-NB15 dataset.

The resulting software artifact—a Streamlit-based GUI with a Hybrid Rule Engine—demonstrates the practical application of this research. It translates complex algorithmic outputs into actionable security intelligence, addressing the core objectives of reducing response times and automating threat triage. While challenges remain in detecting the rarest attack categories, the system represents a significant step forward from legacy signature-based defenses, offering a robust, anomaly-based shield for modern digital infrastructure.

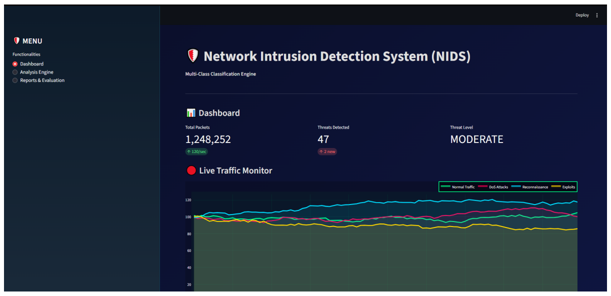
X. GITHUB REPOSITORY

https://github.com/hassam-cheemaa/Network-Intrusion-Detection-System-using-Multiclassification

XI. LINKEDIN POST

https://www.linkedin.com/posts/aadilmasaud_cybersecurity-machinelearning-datascience-activity-740640556783508275

XII. Appendix A: System Diagrams



Network Intrusion Detection System (NIDS)

Multi-Class Classification Engine

Dashboard

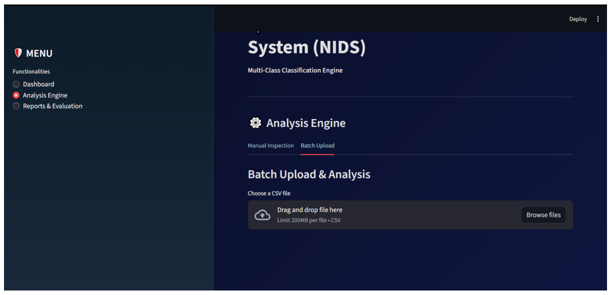
Total Packets: 1,248,252 (↑ 120/sec)

Threats Detected: 47 (↑ 2 new)

Threat Level: MODERATE

Live Traffic Monitor

Fig. 3. Dashboard



System (NIDS)

Multi-Class Classification Engine

Analysis Engine

Batch Upload & Analysis

Choose a CSV file

Drag and drop file here

Limit 200MB per file • CSV

Fig. 4. Batch Processing

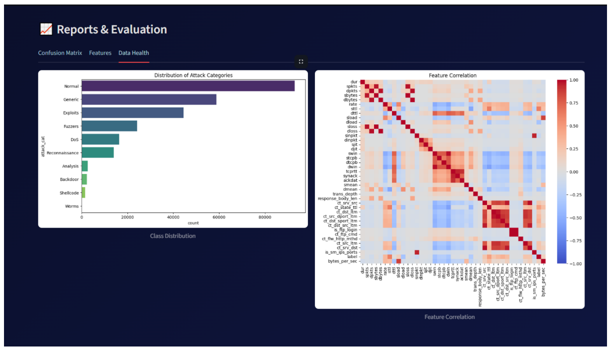
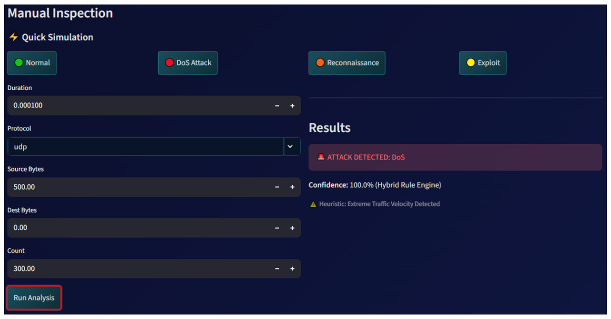


Fig. 5. Analysis



Manual Inspection

Quick Simulation

Duration: 0.000100

Protocol: udp

Source Bytes: 500.00

Dest Bytes: 0.00

Count: 300.00

Results

ATTACK DETECTED: DoS

Confidence: 100.0% (Hybrid Rule Engine)

Heuristic: Extreme Traffic Velocity Detected

Fig. 6. Manual Inspection

References

[1] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," *2015 Military Communications and Information Systems Conference (MilCIS)*, Canberra, ACT, Australia, 2015, pp. 1–6.

[2] N. Moustafa and J. Slay, "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set," *Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 2016.

[3] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.

[4] 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, p. 102419, 2020.

[5] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Advances in Neural Information Processing Systems*, vol. 30, pp. 3146–3154, 2017.

[6] M. Sarhan, S. Layeghy, and M. Portmann, "Standard feature set for network intrusion detection systems," *Big Data and Cognitive Computing*, vol. 4, no. 4, p. 38, 2020.

[7] S. M. Kasongo and Y. Sun, "A deep learning method with wrapper based feature extraction for wireless intrusion detection system," *Computers & Security*, vol. 92, p. 101752, 2020.

[8] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[9] M. A. Alsharaiah, et al., "An Efficient Intrusion Detection System Based on Random Forest and Flower Pollination Optimization Algorithm," *IEEE Access*, vol. 12, pp. 15020–15035, 2024.

[10] N. Gupta and V. Jindal, "Intrusion Detection System using Hybrid Classifiers," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 4, 2022.

[11] D. Zhang, et al., "Network Intrusion Detection Based on LightGBM and Grammatical Evolution," *IEEE Access*, vol. 8, pp. 99300–99312, 2020.

[12] M. R. Islam, et al., "Explainable Artificial Intelligence for Network Intrusion Detection System," *IEEE Access*, vol. 9, pp. 135232–135248, 2021.

[13] S. Al-Otaibi, "Towards an Efficient Feature Selection Method for Network Intrusion Detection Systems," *IEEE Access*, vol. 9, pp. 48674–48685, 2021.

[14] Kurniabudi, et al., "CICIDS2017 Dataset Feature Selection and Imbalanced Handling for Intrusion Detection System," *International Conference on Information and Communications Technology (ICOIACT)*, 2020.