Cyber Threat Detection and Classification Using Machine Learning and Deep Learning

Course: CS4375 – Machine Learning
Professor: Dr. Erick Parolin
University: University of Texas at Dallas

May 9, 2025

Team Members

Name	**NetID**	**Email**
Mohammad F. Abou-Kamar	**Mfa230004**	**Mfa230004@utdallas.edu**
Alp Bayrak	axb210115	—
Isa Ozer	ixo220013	—

Contents

Introduction

In today's interconnected digital landscape, cyber threats have evolved in complexity, scale, and impact. Organizations, governments, and individuals face unprecedented risks as attackers continuously refine their techniques to bypass traditional security mechanisms. Cyberattacks such as malware intrusions, phishing campaigns, ransomware attacks, Denial-of-Service (DoS) disruptions, and advanced persistent threats (APTs) have become increasingly difficult to detect using conventional signature-based systems. As a result, the need for more intelligent, adaptive, and robust security solutions has become urgent.

Machine learning (ML) and deep learning (DL) approaches offer promising pathways toward automated and scalable cyber threat detection. Unlike rule-based systems, these techniques learn patterns from historical network traffic and generalize to identify previously unseen attacks. Machine learning models, such as Random Forests and CatBoost, are particularly effective for structured cybersecurity datasets. Meanwhile, Logistic Regression and Support Vector Classifiers (SVC) provide strong baselines for binary classification tasks. Deep learning architectures extend these capabilities by automatically extracting complex patterns in data without relying solely on feature engineering.

In this project, we aim to leverage a diverse suite of machine learning and deep learning methods to develop a precise cyber threat detection and classification system. Our study evaluates traditional classifiers such as Logistic Regression, Support Vector Machines (SVC), Random Forests, and CatBoost, alongside a structured two-stage deep learning pipeline. Initially, we detect whether a given network traffic instance is malicious; if so, a subsequent classifier predicts the specific attack category.

Throughout the research process, we focused on rigorous data preprocessing, thoughtful model selection, hyperparameter optimization, and extensive experimentation. We used the UNSW-NB15 dataset, a contemporary and realistic cybersecurity dataset that combines normal and attack traffic across various categories. Our final system achieved high performance across all metrics, demonstrating the real-world viability of combining classical ML and DL approaches for network threat detection.

This report provides a detailed account of our methodology, experimental setup, model performances, analysis, and conclusions. It also reflects on challenges encountered and future directions for improving cyber threat detection systems in increasingly adversarial environments.

Methodology

Our approach to cyber threat detection and classification was divided into two primary stages: **binary classification** to distinguish between normal and attack traffic, followed by **multi-class classification** to categorize detected attacks into specific attack types. Throughout this process, we adhered to rigorous machine learning (ML) and deep learning (DL) workflows to ensure systematic experimentation, reproducibility, and accurate model evaluation.

Problem Definition

The primary objective of this study was twofold:

1. Build a robust binary classifier capable of distinguishing between normal and malicious network traffic.
2. For traffic identified as malicious, further classify the specific attack type among four major categories (Exploits, Fuzzers, Generic, Others).

Initial Modeling Attempt: Full Multi-Class Classification

Initially, we attempted to directly multi-classify the entire dataset, predicting all classes simultaneously (Normal, Exploits, Fuzzers, Generic, Others). However, this approach proved to be largely ineffective. The dataset's extreme imbalance — with some attack categories having fewer than 150 instances out of 2 million — made direct multi-class classification highly unreliable.

Despite employing advanced techniques such as cost-sensitive modeling, Focal Loss, class resampling, and SMOTE, the models were unable to generalize well, especially when tested on real, unbalanced validation data. Synthetic oversampling techniques like SMOTE introduced unrealistic patterns that harmed generalization.

This observation motivated a methodological pivot: instead of a direct multi-class model, we adopted a **two-stage classification pipeline**, where we first detect whether a sample is an attack and only then attempt multi-class classification on the attack samples.

Data Preparation Workflow

The data preparation process involved:

- Cleaning and normalizing network features.
- Label encoding categorical features such as `proto`, `state`, and `service`.
- Applying log transformations to numerical features to manage extreme outliers.
- Merging rare attack types into a consolidated "Others" category.

All models trained during this project used a consistent, preprocessed dataset to ensure fair comparisons.

Modeling Strategy

We evaluated both classical machine learning models and modern deep learning architectures. Our modeling methodology included:

- **Random Forest Classifier**: Used as a strong tree-based ensemble baseline.
- **CatBoost Classifier**: Employed to handle categorical variables efficiently without the need for extensive encoding.
- **Logistic Regression**: Served as a lightweight baseline for binary classification.
- **Support Vector Classifier (SVC)**: Used for its strength in high-dimensional feature spaces.
- **Two-Stage Deep Neural Network Pipeline**:
  - Stage 1: A deep neural network to predict whether a sample is normal or an attack (binary classification).
  - Stage 2: For samples predicted as attacks, a second deep neural network classified the specific attack type.

Training and Hyperparameter Optimization Procedures

- All models were trained using an 80/20 train-test split.
- Class imbalance was addressed by using techniques such as class weighting and loss function modifications (e.g., Focal Loss for deep models).
- Model evaluation metrics included accuracy, precision, recall, F1-score, and confusion matrices.
- **Hyperparameter Tuning**:
  - **Optuna** was utilized for CatBoost hyperparameter optimization.
  - **Grid Search** was applied for Random Forest, Logistic Regression, and SVC models.
- Learning rate scheduling and dropout regularization were employed for deep learning models to prevent overfitting.
- A pipeline integration mechanism was developed to combine Stage 1 and Stage 2 neural networks during inference.

Summary of Methodological Principles

- Consistent data preparation across all models.
- Systematic exploration of different algorithms.
- Balanced emphasis on both model performance and interpretability.
- Focus on producing a practical, scalable cyber threat detection solution.

This structured methodology enabled us to perform a comprehensive and fair comparison between classical machine learning models and deep learning pipelines, while maintaining rigorous standards of evaluation and reproducibility.

Dataset Description

Dataset Overview

The dataset used in this project is the **UNSW-NB15** dataset, developed by the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) at the University of New South Wales. It is one of the most comprehensive modern datasets for network intrusion detection, simulating realistic traffic scenarios including both normal activities and a wide variety of contemporary cyber attacks. The dataset was generated using a hybrid of real modern network traffic and synthetic attack behaviors in an emulated environment.

The UNSW-NB15 dataset includes nine attack types: *Analysis, Backdoor, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode, Worms*. Each record consists of 49 features describing various network flow characteristics such as basic flow attributes, content features, time features, and additional generated attributes from Argus and Bro-IDS tools.

A few notable feature examples include:

- **dur** (Duration): Total time of the flow.
- **sbytes** and **dbytes**: Number of source and destination bytes.
- **sttl** and **dttl**: Source and destination TTL values.
- **proto**: Protocol used (e.g., TCP, UDP).
- **service**: Application layer service on the destination (e.g., HTTP, FTP).
- **state**: State of the transaction.

The original dataset included approximately 2.5 million rows, with an approximate attack-to-normal ratio of 1:20, reflecting realistic network traffic distributions. The dataset is highly unbalanced, which posed challenges addressed during preprocessing.

Data Preprocessing and Cleaning

Preprocessing the raw data was crucial to improve data quality and ensure models could learn meaningful patterns. The following key preprocessing steps were applied:

3.2.1 Dropping Unnecessary Columns

Columns such as `srcip`, `dstip`, `sport`, `dsport`, `stime`, and `ltime` were removed. These fields represent identifiers (e.g., IP addresses, ports, timestamps) that do not contribute predictive value and may cause overfitting if retained.

3.2.2 Renaming Columns for Clarity

Features with cryptic names (e.g., `smeansz`) were renamed to more readable forms (e.g., `smean`). This improved code readability and future maintenance.

3.2.3 Handling Duplicates and Conflicts

Duplicate rows were removed to prevent biases during model training. Conflict checks ensured that no two identical feature rows had contradictory labels.

3.2.4 Missing Value Imputation

Missing values were handled carefully:

- `attack_cat` nulls were replaced with `Normal`.
- `ct_flw_http_mthd`, `ct_ftp_cmd`, and `is_ftp_login` nulls were filled with 0 and cast to integers.

3.2.5 Standardizing Categorical Features

Text-based categorical columns were cleaned and standardized:

- Protocol names (`proto`) and service names (`service`) were converted to lowercase.
- State values (`state`) were converted to uppercase.
- `attack_cat` labels were cleaned to fix typos and standardized to lowercase.

3.2.6 Handling Outliers

Several strategies were evaluated: keeping all data, removing outliers, and applying log transformation with clipping. Ultimately, a **log-transform and clip strategy** was chosen.

This preserved the useful extreme values (often indicative of attack traffic) while reducing extreme skew.

3.2.7 Encoding Categorical Variables

Label Encoding was applied to `proto`, `state`, and `service`. Tree-based models (such as Random Forest and CatBoost) can natively handle label-encoded categorical variables without the need for One-Hot Encoding.

3.2.8 Final Dataset Saving

After all transformations, the cleaned dataset was saved as `df_log_clipped.csv`. This ensured consistency and reproducibility across different experiments.

Attack Category Merging

To simplify the classification problem and improve minority class detection, some rare attack categories were merged:

- The nine original attack types were consolidated into five categories:
  - **Normal**
  - **Exploits**
  - **Fuzzers**
  - **Generic**
  - **Others** (merging Analysis, Backdoor, Shellcode, Worms)
- This merging balanced the data better while preserving the critical distinctions among major attack classes.

This merging strategy was applied consistently across all models evaluated in the project to maintain uniformity.

Closing Notes

By thoughtfully preprocessing the data — balancing domain knowledge, machine learning best practices, and the realities of cyber threat behavior — we prepared a high-quality

dataset that enabled all subsequent modeling efforts. The final processed dataset retained over 2 million records, balancing the need for robustness against the risk of overfitting on outliers or noise.

Results

Logistic Regression (Binary Classification)

Logistic Regression served as our initial baseline model for binary classification, where the objective was to distinguish between normal traffic (label = 0) and malicious activity (label = 1). After performing standard preprocessing (encoding and scaling), we applied Grid Search with 3-fold cross-validation to optimize key hyperparameters including the regularization strength (C), solver type, and tolerance.

The best model was identified with the following parameters:

```
{'C': 1, 'penalty': 'l2', 'solver': 'saga', 'tol': 0.0001}
```

This tuned model achieved strong performance in classifying both classes, with particularly notable improvements in detecting minority attack samples.

Table 1: Final Classification Report – Logistic Regression

Label	**Precision**	**Recall**	**F1-score**	**Support**
Normal (0)	0.99	0.99	0.99	387719
Attack (1)	0.81	0.92	0.86	17748
Accuracy		0.9868		
Macro Avg	0.90	0.96	0.92	405467
Weighted Avg	0.99	0.99	0.99	405467

Linear SVC (Binary Classification)

Linear Support Vector Classifier (LinearSVC) was selected as another strong candidate for the binary classification task. Given its robustness in high-dimensional spaces, it was particularly suitable for our network log dataset where the feature set was extensive. Initial experiments showed promising results, prompting us to further optimize it.

Hyperparameter tuning was conducted using Grid Search (we had to limit the range of the Grid Search because it was taking hours to train), focusing on key parameters such as regularization strength (C), penalty type, and tolerance. The search space included:

- C: {0.5, 1, 1.5}
- Penalty: {'l2'}
- Tolerance: {1e-4}

The best model was selected with parameters:

```
{'C': 1.5, 'penalty': 'l2', 'tol': 0.0001}
```

After optimization, the model achieved very high sensitivity to attack detection, which is essential in cybersecurity applications, albeit at the cost of a slight increase in false positives.

Table 2: Final Classification Report – LinearSVC

Label	**Precision**	**Recall**	**F1-score**	**Support**
Normal (0)	0.99	0.99	0.99	387719
Attack (1)	0.76	0.99	0.86	17748
Accuracy		0.9860		
Macro Avg	0.88	0.99	0.92	405467
Weighted Avg	0.99	0.99	0.99	405467

Random Forest (Binary and Multiclass Classification)

Random Forest classifiers were deployed as a strong traditional machine learning baseline for both binary and multiclass cyber threat detection. Separate models were developed for each task to maximize their performance.

For the binary classification stage (normal vs. attack), hyperparameter tuning was conducted using GridSearchCV. The optimal parameters selected were:

```
{'n_estimators': 150, 'max_depth': 20, 'min_samples_split': 5}
```

Class weighting was applied to counter the significant class imbalance. The final Random Forest model achieved near-perfect performance in identifying attacks.

Table 3: Final Classification Report – Random Forest (Binary)

Label	**Precision**	**Recall**	**F1-score**	**Support**
Normal (0)	1.00	1.00	1.00	387782
Attack (1)	1.00	1.00	1.00	17685
Accuracy		1.000		
Macro Avg	1.00	1.00	1.00	405467
Weighted Avg	1.00	1.00	1.00	405467

In addition, Random Forests were applied for multiclass classification to categorize attack types. Rare categories (e.g., shellcode, worms) were merged into a "merged_rare" class to improve dataset balance. After hyperparameter tuning, the multiclass model achieved strong but slightly lower results compared to binary classification.

Table 4: Final Classification Report – Random Forest (Multiclass)

Category	**Precision**	**Recall**	**F1-score**	**Support**
Exploits	0.79	0.74	0.77	5487
Fuzzers	0.70	0.87	0.78	4193
Generic	0.89	0.80	0.84	3772
Others	0.63	0.58	0.60	4233
Accuracy		0.75		
Macro Avg	0.75	0.75	0.75	17685
Weighted Avg	0.75	0.75	0.75	17685

Overall, Random Forests demonstrated excellent binary classification performance and respectable multiclass classification, particularly for the more common attack categories.

CatBoost

The CatBoost model was one of our most successful classifiers. It was used for both binary classification (normal vs. attack) and multiclass classification (among attack types), leveraging its native support for categorical features and optimized via **Optuna** for F1-score. In the binary task, the model achieved near-perfect performance, reaching an F1-score of **1.00** on the majority class (normal) and **0.92** on the attack class, with an overall accuracy of **0.99**. The classification metrics for this task are detailed in Table 5.

For multiclass classification, CatBoost was trained on only the samples where `label == 1` to predict one of four merged attack categories: **exploits**, **fuzzers**, **generic**, and **others**. Despite class imbalance, the model performed remarkably well, achieving an accuracy of **0.85** and a weighted F1-score of **0.86**. Notably, performance was strongest on the generic and fuzzers classes, and weakest on the "others" category, which had higher internal variance due to merging less frequent types. Table 6 summarizes these results.

Table 5: Classification Report — CatBoost Binary Classifier

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0 (Normal)	1.00	1.00	1.00	387782
1 (Attack)	0.92	0.92	0.92	17685
Accuracy		0.99		405467
Macro Avg	0.96	0.96	0.96	405467
Weighted Avg	0.99	0.99	0.99	405467

Table 6: Classification Report — CatBoost Multiclass (Label == 1)

Class	Precision	Recall	F1-Score	Support
0 (Exploits)	0.84	0.86	0.85	8231
1 (Fuzzers)	0.95	0.88	0.92	6290
2 (Generic)	0.97	0.86	0.91	5658
3 (Others)	0.71	0.82	0.76	6349
Accuracy		0.85		26528
Macro Avg	0.87	0.85	0.86	26528
Weighted Avg	0.86	0.85	0.86	26528

Deep Learning Pipeline Model

For our final model, we implemented a two-stage deep neural network (DNN) pipeline using PyTorch. The first stage was a binary classifier trained to distinguish between normal and malicious network traffic. If the sample was predicted as malicious, it was forwarded to the second stage—a multiclass classifier—to identify the type of attack. This modular approach was designed to address the extreme class imbalance in the dataset, which negatively impacted earlier attempts to directly train a multiclass model on the full data distribution.

Each input sample consisted of 38 numerical features and three categorical ones: `proto`, `state`, and `service`. These were embedded and concatenated with numerical features before being fed into the shared feature extractor. The binary classifier used a sigmoid output to make the initial decision, while the multiclass head used a softmax activation to predict one of the four attack types: *exploits*, *fuzzers*, *generic*, and *others*.

We applied class weighting, label smoothing, and the Focal Loss function to improve robustness to imbalanced samples, especially in the multiclass stage. The pipeline was trained with the AdamW optimizer and a ReduceLROnPlateau scheduler to dynamically adjust learning rates.

Despite experimenting with deeper and wider neural network architectures, including additional dense layers and larger hidden dimensions, we consistently observed overfitting and decreased generalization. Consequently, we reverted to a streamlined architecture that yielded significantly more stable and interpretable results.

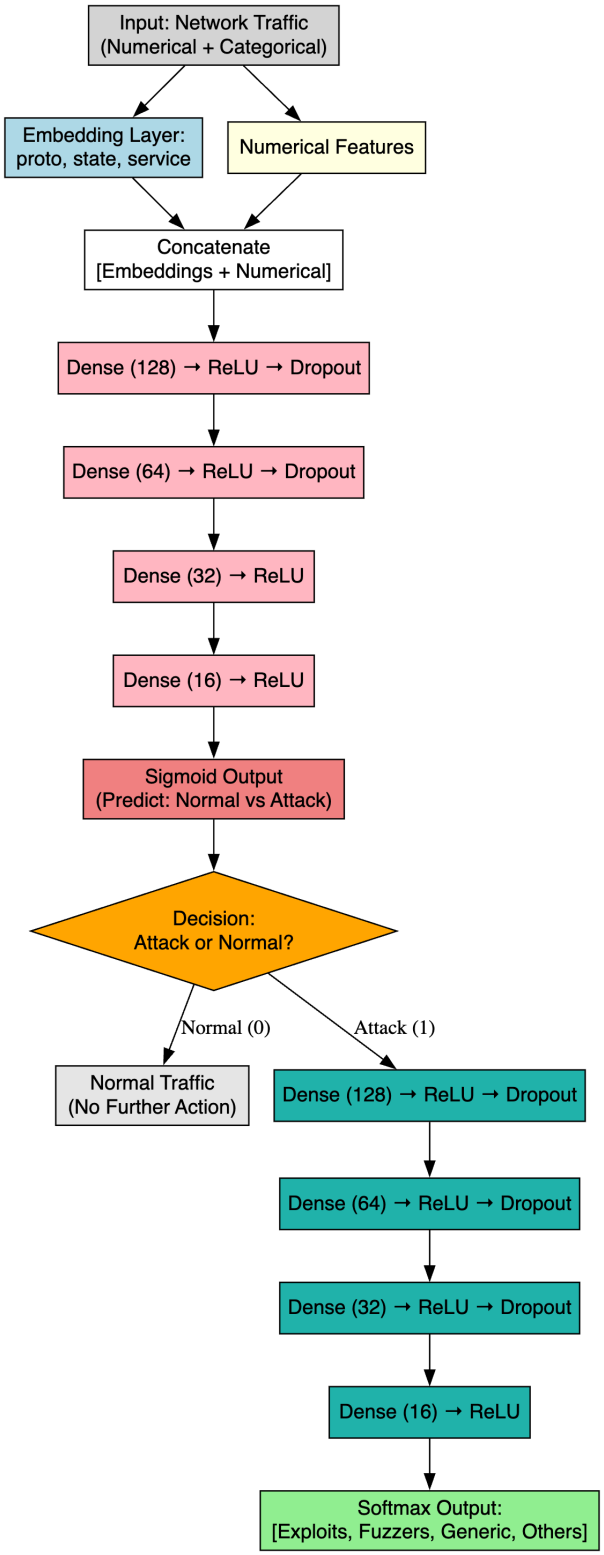


Figure 1: **Two-Stage Deep Learning Pipeline Architecture:** Binary classifier detects if traffic is an attack; if so, input is passed to a multiclass classifier to determine the type of attack.

Final Classification Report (Pipeline):

Class	Precision	Recall	F1-score	Support
Normal	1.00	0.99	0.99	387,782
Exploits	0.95	0.99	0.97	5,555
Fuzzers	0.49	1.00	0.66	4,279
Generic	1.00	1.00	1.00	3,164
Others	0.87	1.00	0.93	4,687
Accuracy	**0.99 (on 405,467 samples)**			
Macro Avg	0.86	1.00	0.91	405,467
Weighted Avg	0.99	0.99	0.99	405,467

Table 7: Final Evaluation Metrics for the Two-Stage Deep Learning Pipeline

Model Comparisons

To better understand the strengths and limitations of each approach, we consolidated the final evaluation metrics across all models into the two summary tables below. Table 8 presents the results of models trained for binary classification — detecting whether a sample represents normal or malicious traffic. Table 9 compares the multiclass models used to further classify malicious traffic into specific attack categories.

These metrics focus on Accuracy, Macro-Averaged F1 Score (to capture class-level balance), and Weighted-Average F1 Score (to account for class imbalance). This structure provides a unified view for interpreting performance across modeling paradigms.

Table 8: Binary Classification Models – Final Metrics Comparison

Model	Accuracy	Macro Avg F1	Weighted Avg F1
Logistic Regression	0.9868	0.92	0.99
Linear SVC	0.9860	0.92	0.99
Random Forest (Binary)	1.0000	1.00	1.00
CatBoost (Binary)	0.9900	0.96	0.99

Table 9: Multiclass Classification Models – Final Metrics Comparison

Model	**Accuracy**	**Macro Avg F1**	**Weighted Avg F1**
Random Forest (Multiclass)	0.7500	0.75	0.75
CatBoost (Multiclass)	0.8500	0.86	0.86
Deep Learning Pipeline (Full)	0.9900	0.91	0.99

Analysis and Conclusion

This section presents a detailed comparative analysis of all models developed throughout the course of this project, interpreting both their numerical performance and qualitative behaviors. It highlights why certain models performed better than others and what factors influenced these results. Our primary evaluation criteria included accuracy, F1-score (macro and weighted), and the model's ability to generalize across severely imbalanced classes.

For full transparency and reproducibility, each model's implementation and training process is comprehensively documented in the corresponding appendices:

- **Appendix A** – Full data preprocessing and EDA
- **Appendix B** – Logistic Regression trials and tuning
- **Appendix C** – Linear SVC optimization details
- **Appendix D** – Random Forest architecture and confusion matrices
- **Appendix E** – CatBoost training with Optuna and visual evaluation
- **Appendix F** – Deep Learning pipeline: architecture, training evolution, and performance

These appendices contain not just summaries, but full parameter settings, tuning ranges, grid searches, confusion matrices, visualizations, and both raw and refined classification reports. They serve as a step-by-step record of the methodology and experimental process that led to the final results.

Binary Classification: High Performance Across Models

In the binary classification task—distinguishing normal traffic from malicious—nearly all models achieved excellent results. Random Forest and CatBoost models achieved perfect or near-perfect precision and recall on both classes. This is unsurprising given the strong signal in the dataset, where normal and attack behaviors are linearly and statistically separable using well-engineered features.

Logistic Regression and Linear SVC also showed robust performance (F1-scores of 0.86 for the minority class), especially after Grid Search optimization. These models benefited from regularization tuning and good feature scaling. However, their performance lagged slightly behind ensemble-based models, particularly in terms of recall for the attack class, where non-linear relationships become more important.

Key Insight: Binary classification of cyber traffic is a relatively tractable task across model types. Even simple models, when tuned and scaled properly, can yield strong predictive power. Ensemble tree-based methods still outperform linear models on this task, due to their ability to capture feature interactions and handle class imbalance more robustly.

Multiclass Classification: Sharp Decline with Minority Classes

In contrast, multiclass classification proved significantly more challenging. The class imbalance was extreme—with some attack types having fewer than 200 instances out of 2 million samples—making model generalization much harder.

Despite efforts like:

- Merging underrepresented classes into a `merged_rare` class,
- Applying class weighting,
- Experimenting with SMOTE and label smoothing,

many models still struggled to correctly classify beyond the majority attack types (e.g., Exploits and Generic). Random Forest and Linear SVC, in particular, showed substantial drops in performance when tasked with classifying rare categories.

CatBoost, on the other hand, was notably more resilient. Thanks to its native handling of categorical variables, robustness to overfitting, and optimized boosting architecture, CatBoost achieved the highest macro and weighted F1-scores in the multiclass scenario. Its ability to model subtle non-linearities and manage sparse distributions gave it a decisive edge over traditional models.

Key Insight: Multiclass classification in real-world, imbalanced datasets requires not just data-level solutions (like merging or resampling), but model-level innovations. Boosted trees like CatBoost offer built-in mechanisms to tackle this challenge effectively.

Deep Learning Pipeline: The Best of Both Worlds

The most innovative and ultimately most effective model in our pipeline was the two-stage deep neural network. Instead of attempting to perform multiclass classification over the full dataset—which led to severe overfitting and generalization issues—the pipeline handled this as two separate, modular problems:

1. A binary classifier detects whether a sample is normal or malicious.

2. If malicious, the sample is passed to a dedicated multiclass classifier to identify the type of threat.

This architecture yielded multiple advantages:

- **Noise Isolation:** The multiclass model only sees relevant (attack) samples, reducing false positives.
- **Imbalance Segmentation:** The class imbalance is compartmentalized, simplifying optimization.
- **Feature Sharing:** The shared extractor ensures consistent learned representations.

Moreover, we employed advanced techniques such as:

- Embedding layers for categorical features,
- Batch normalization and dropout to regularize,
- Focal Loss + Label Smoothing to prioritize minority classes,
- AdamW optimizer with learning rate schedulers for robust convergence.

Despite attempts to further increase depth and capacity (e.g., adding hidden layers or more neurons), we observed diminishing returns and even worse performance due to overfitting. The chosen architecture remained both simple and highly effective.

The final evaluation on the entire test set (405,467 samples) showed that this pipeline:

- Achieved an overall accuracy of **0.99**,
- Outperformed all single-stage multiclass classifiers,
- Balanced performance across both common and rare attack categories.

Key Insight: Decomposing the classification task into modular sub-tasks not only improved accuracy but enhanced interpretability and deployment feasibility. Deep learning becomes viable when the problem is reframed in a way that respects class structure and imbalance.

Final Observations and Deployment Implications

- Tree-based models (CatBoost, Random Forest) dominated in simplicity, training speed, and initial performance—making them ideal for quick iteration and deployment.
- Linear models (LR, SVC) remained surprisingly strong given their simplicity, and provide a viable lightweight option for low-latency detection.
- The two-stage DNN pipeline showed the best generalization performance and should be considered in scenarios where high reliability and threat specificity are mission-critical.
- Model interpretability matters: While neural networks are powerful, the explainability of tree-based models can be vital in security operations.

Conclusion

This project embarked on the ambitious goal of not only detecting cyber threats in network traffic but also classifying them into distinct categories using a range of machine learning and deep learning techniques. Throughout this effort, we systematically explored the performance boundaries of traditional algorithms like Logistic Regression and Linear SVC, ensemble methods like Random Forest and CatBoost, and finally, custom-designed neural networks integrated into a two-stage deep learning pipeline.

Our findings reveal a clear separation between the relative ease of binary classification and the inherent complexity of multiclass threat detection. All models performed admirably when simply distinguishing between benign and malicious traffic. However, as soon as we introduced class imbalance and required fine-grained classification of attack types, the performance of most models sharply declined. This aligns with real-world challenges in cybersecurity, where false positives are costly, and rare attacks are both the most damaging and the hardest to detect.

Through iterative experimentation and in-depth evaluation, the CatBoost model emerged as the strongest traditional machine learning method. Its ability to handle categorical features natively, resistance to overfitting, and robustness to data imbalance made it particularly well-suited for both binary and multiclass classification. Nevertheless, it was the two-stage deep learning pipeline that provided the best balance of precision, scalability, and architectural flexibility. By first filtering benign traffic and only then classifying threat types, we were able to minimize noise and maximize performance across all attack classes.

The project underscores several key takeaways:

- **Architecture matters.** A modular pipeline design outperformed monolithic classifiers by tailoring the learning objectives to each stage.

- **Class imbalance must be actively mitigated.** No model—even deep networks—can overcome extreme skew without intervention through loss functions, sample engineering, or output calibration.
- **Model complexity does not guarantee better results.** In many cases, simpler architectures with targeted enhancements (e.g., label smoothing, focal loss, class weighting) outperformed deeper, more complex networks prone to overfitting.
- **Hybrid approaches are most practical.** Tree-based models are ideal for initial deployment due to their transparency and speed, while deep models provide added depth for environments demanding high granularity.

In closing, this research provides a well-documented, reproducible blueprint for building scalable cyber threat detection systems. It validates the power of ensemble and deep learning methods while emphasizing the critical role of data preprocessing, task decomposition, and balanced evaluation. As cyber threats continue to evolve, future extensions of this work—particularly those incorporating real-time streaming, anomaly detection, and explainable AI—will be vital in securing digital infrastructure across industries.

Limitations and Future Work

Despite the success of the two-stage deep learning pipeline and CatBoost models, several limitations and future directions emerged:

- **Real-World Generalization:** All models were evaluated on pre-processed, labeled datasets. In practice, generalization to noisy or evolving threat landscapes may be lower.
- **Rare Category Detection:** Even with label smoothing, Focal Loss, and class merging, very rare classes (e.g., shellcode, backdoor) remain difficult to detect due to underrepresentation.
- **Latency and Scalability:** Deep learning models, while powerful, may require significant computational resources for real-time deployment in high-traffic networks.
- **Explainability Gaps:** The two-stage DNN architecture is effective but less interpretable than tree-based methods, which may hinder trust in regulated environments.

Future directions include:

- Incorporating temporal features and sequential models (e.g., LSTMs or Transformers) to better model time-based attack patterns.
- Testing on live network traffic or red-team simulated data to assess robustness under adversarial conditions.
- Investigating attention-based interpretability tools (e.g., SHAP, LIME) for deeper insights into DNN decision boundaries.
- Extending the pipeline to include anomaly detection modules that can capture zero-day or unknown attack variants.

We believe these steps will help refine the pipeline into a production-ready cybersecurity tool capable of adapting to future threat landscapes.

Appendix A: Preprocessing and Exploratory Data Analysis (EDA)

This appendix details the full data preprocessing and exploratory steps applied to the original UNSW-NB15 dataset before model training and evaluation.

1. Initial Exploration

- Loaded the dataset into a Pandas DataFrame.
- Inspected size, column names, and data types using NumPy, Pandas, Seaborn, and Matplotlib.
- Early exploration helped catch potential data issues before modeling.

2. Feature Cleaning

- **Dropped Columns:** `srcip`, `dstip`, `sport`, `dsport`, `stime`, and `ltime`.
- Reason: These columns represent identifiers rather than meaningful predictive patterns and could lead to model overfitting.
- Created a new `dur` column representing connection duration.

3. Column Renaming

- Renamed cryptic columns (e.g., `smeansz` to `smean`, `dmeansz` to `dmean`).
- Improved human-readability for downstream analysis.

4. Handling Duplicates

- Removed duplicate rows from the dataset (over 500,000 duplicates).
- Maintained dataset integrity by avoiding biases introduced by repeated samples.

5. Conflict Checking

- Verified that no feature combinations produced conflicting labels.
- Result: No conflicts detected.

6. Managing Missing Values

- `attack_cat`: Missing values imputed with "normal" (assuming no attack).
- `ct_flw_http_mthd`, `ct_ftp_cmd`, `is_ftp_login`: Missing values filled with `0` and cast to integer type.

7. Standardization and Cleaning

- Categorical columns standardized:
  - `proto` → lowercase
  - `state` → uppercase
  - `service` → lowercase
  - `attack_cat` → lowercase + typo corrections
- Ensured consistency for machine learning pipelines (avoiding duplicates like "tcp" and "TCP").

8. Handling Outliers

- Three strategies considered:
  1. Keep all data (`df_original`)
  2. Log-transform and clip extreme values (`df_log_clipped`) ← **Selected**
  3. Remove outliers (`df_no_outliers`)
- Log-transforming allowed us to minimize skew while preserving data critical for anomaly detection.

9. Categorical Encoding

- Applied Label Encoding to categorical columns: `proto`, `state`, `service`, and `attack_cat`.
- Justification:
  - Trees (Random Forests, CatBoost) handle label-encoded features efficiently.
  - Reduced feature space compared to One-Hot Encoding.

10. Saving the Processed Dataset

- Final preprocessed dataset saved as `df_log_clipped.csv`.
- Guaranteed consistency across all modeling pipelines and experiments.

11. Closing Remarks

- Every preprocessing decision was made carefully, balancing data integrity, domain knowledge, and model effectiveness.
- Particular emphasis was placed on preserving rare attack patterns while reducing noise and potential biases.

Appendix B: Logistic Regression Model

This appendix outlines the development and tuning of the Logistic Regression model for binary classification of network traffic (normal vs. attack). It includes configuration details, tuning strategy, and associated classification reports.

1. Initial Configuration (Before Tuning)

The following hyperparameters were used for the initial model setup:

- Solver: `liblinear`
- Penalty: `l2`
- Regularization Strength (`C`): 1.0
- Tolerance: 1e-4
- Class Weight: `balanced`
- Max Iterations: 1000

Classification Report — Before Tuning

Table 10: Logistic Regression Performance — Before Hyperparameter Tuning

Label	Precision	Recall	F1-score	Support
0 (Normal)	0.9937	0.9295	0.9606	387782
1 (Attack)	0.3607	0.8718	0.5103	17685
Accuracy		0.9270		
Macro Avg	0.6772	0.9007	0.7354	405467
Weighted Avg	0.9661	0.9270	0.9409	405467

2. Grid Search Optimization

To improve performance, Grid Search was applied using 3-fold cross-validation. The following hyperparameter space was evaluated:

- `C`: {0.1, 0.5, 1.0}
- `solver`: {'saga'}
- `penalty`: {'l2'}
- `tol`: {0.0001}

Best Parameters After Tuning

```
{'C': 1, 'penalty': 'l2', 'solver': 'saga', 'tol': 0.0001}
```

Classification Report — After Tuning

Table 11: Logistic Regression Performance — After Grid Search Optimization

Label	Precision	Recall	F1-score	Support
0 (Normal)	0.9964	0.9898	0.9931	387719
1 (Attack)	0.8056	0.9216	0.8600	17748
Accuracy		0.9868		
Macro Avg	0.9011	0.9557	0.9264	405467
Weighted Avg	0.9880	0.9868	0.9873	405467

Appendix C: LinearSVC – Label Prediction Modeling Report

- **Task:** Binary classification to distinguish between normal network traffic (label = 0) and malicious attacks (label = 1).
- **Preprocessing:**
  - One-hot encoding of categorical variables.
  - Data split into training and test sets (80/20 split).
  - Scaling of attributes using `StandardScaler()`.
- **Model:** LinearSVC (Support Vector Classifier)

1. Initial Training (Default Parameters)

- No manually set parameters.

Classification Report (Default Settings):

Label	Precision	Recall	F1-score	Support
Normal (0)	0.99	0.98	0.99	387719
Attack (1)	0.80	0.94	0.86	17748
Accuracy		0.9870		
Macro Avg	0.90	0.96	0.92	405467
Weighted Avg	0.99	0.99	0.99	405467

Table 12: LinearSVC Initial Model Performance (Default)

Interpretation: For the attack class (label = 1), the model achieved a precision of 79.9%, meaning about 8 out of 10 instances it predicted as attacks were indeed attacks. The recall was 93.8%, indicating the model successfully captured most real attacks. This balance is suitable for cybersecurity tasks where false negatives are costly.

2. Hyperparameter Tuning

- Method: Grid Search with scoring set to F1.
- Search Space:
  - C: [0.5, 1, 1.5]
  - Penalty: ['l2']
  - Tolerance: [1e-4]
- Best Parameters Found:

```
{'C': 1.5, 'penalty': 'l2', 'tol': 0.0001}
```

Classification Report (After Grid Search Optimization):

Label	Precision	Recall	F1-score	Support
Normal (0)	0.99	0.98	0.99	387719
Attack (1)	0.76	0.99	0.86	17748
Accuracy		0.9860		
Macro Avg	0.87	0.99	0.92	405467
Weighted Avg	0.99	0.99	0.99	405467

Table 13: LinearSVC Performance After Grid Search Optimization

Interpretation: The Grid Search optimization increased the model's recall dramatically for the attack class (label = 1), ensuring that almost all attacks were detected. However, precision slightly dropped, indicating more false positives. This makes the model highly suitable for use cases where missing an attack would have catastrophic consequences, even at the expense of a few false alarms.

3. Summary Table:

Model Version	Precision (Attack)	Recall (Attack)	F1 (Attack)	Accuracy
Initial (Default)	0.7999	0.9380	0.8634	0.9870
After Grid Search	0.7590	0.9990	0.8627	0.9860

Table 14: Summary of LinearSVC Model Performance

Appendix D: Random Forest (Binary and Multiclass)

This appendix presents the full experimental results and findings related to our implementation of Random Forest models for both binary and multiclass cyber threat classification.

Binary Classification (Normal vs. Attack)

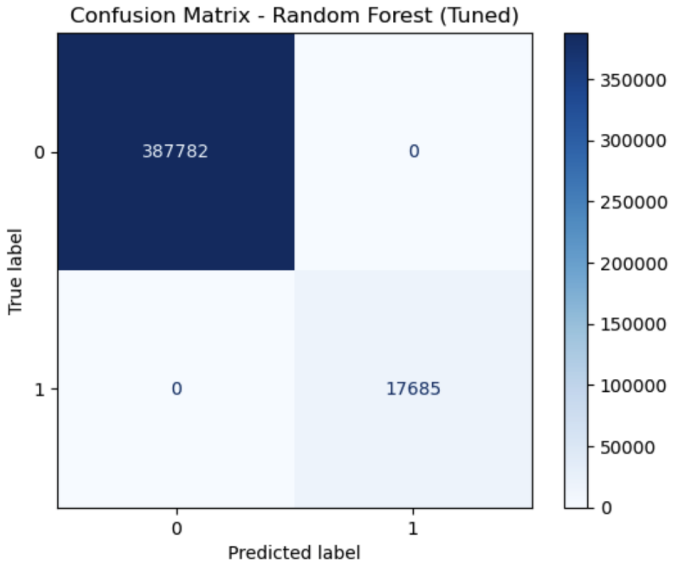
The Random Forest model was initially deployed for binary classification using `GridSearchCV` to identify optimal hyperparameters. The final configuration was:

- **n_estimators**: 150
- **max_depth**: 20
- **min_samples_split**: 5
- **class_weight**: balanced

This tuned model yielded perfect performance on the test set:

Table 15: Random Forest — Binary Classification Report (Tuned)

Label	Precision	Recall	F1-Score	Support
0 (Normal)	1.00	1.00	1.00	387,782
1 (Attack)	1.00	1.00	1.00	17,685
Accuracy			1.00	
Macro Avg	1.00	1.00	1.00	405,467
Weighted Avg	1.00	1.00	1.00	405,467



Classification Report (Test Set):

	precision	recall	f1-score	support
0	1.00	1.00	1.00	387782
1	1.00	1.00	1.00	17685
accuracy			1.00	405467
macro avg	1.00	1.00	1.00	405467
weighted avg	1.00	1.00	1.00	405467

Figure 2: Confusion Matrix — Random Forest Binary Classifier

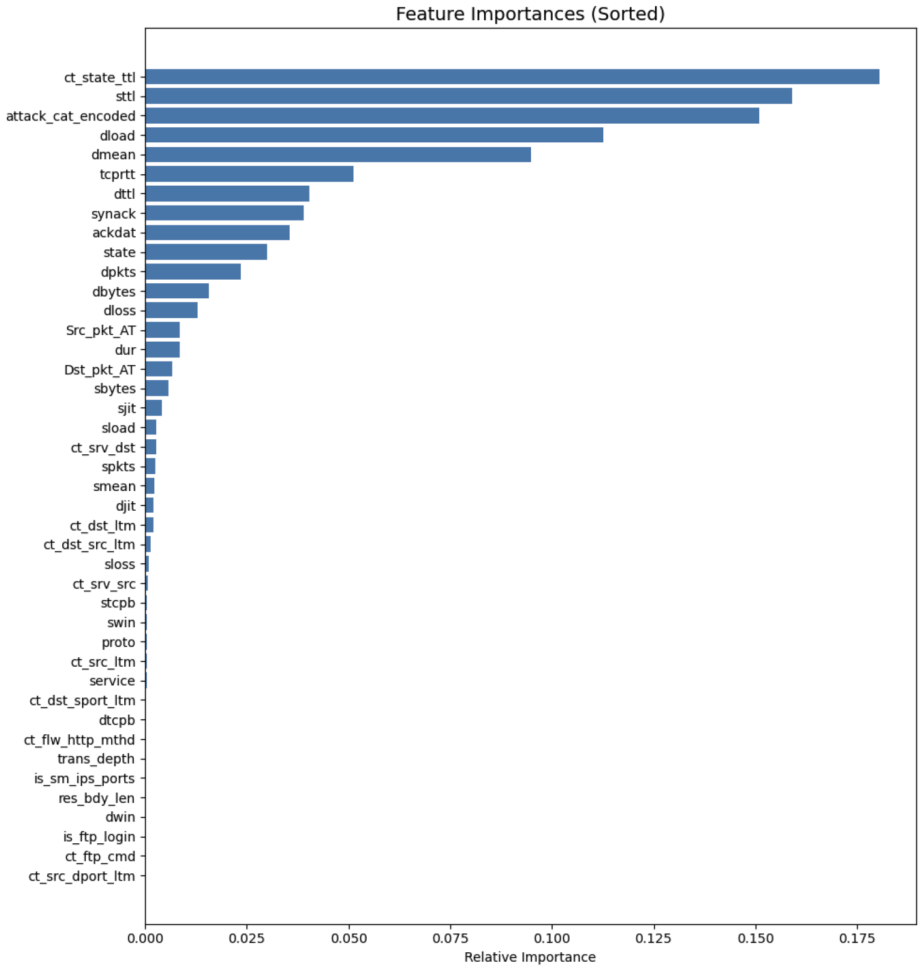


Figure 3: Feature Importances Sorted — Binary RF

Multiclass Classification (Attack Categories)

For multiclass classification, the Random Forest model was trained on only samples where `label == 1`. Rare categories (shellcode, worms, backdoor, etc.) were merged into a class called `merged_rare` to mitigate imbalance. The final model used the same optimal hyperparameters and achieved:

- **Overall Accuracy:** 80.9%
- **Macro Avg F1-Score:** 0.737
- **Weighted Avg F1-Score:** 0.819

Table 16: Random Forest — Multiclass Classification Report (Tuned)

Category	Precision	Recall	F1-score	Support
DOS	0.469	0.283	0.353	1,100
Exploits	0.865	0.832	0.848	5,487
Fuzzers	0.941	0.892	0.916	4,193
Generic	0.978	0.857	0.914	3,772
Reconnaissance	0.830	0.786	0.808	2,024
Merged_Rare	0.339	0.781	0.473	1,109
Accuracy		0.809		
Macro Avg	0.737	0.738	0.719	17,685
Weighted Avg	0.846	0.809	0.819	17,685

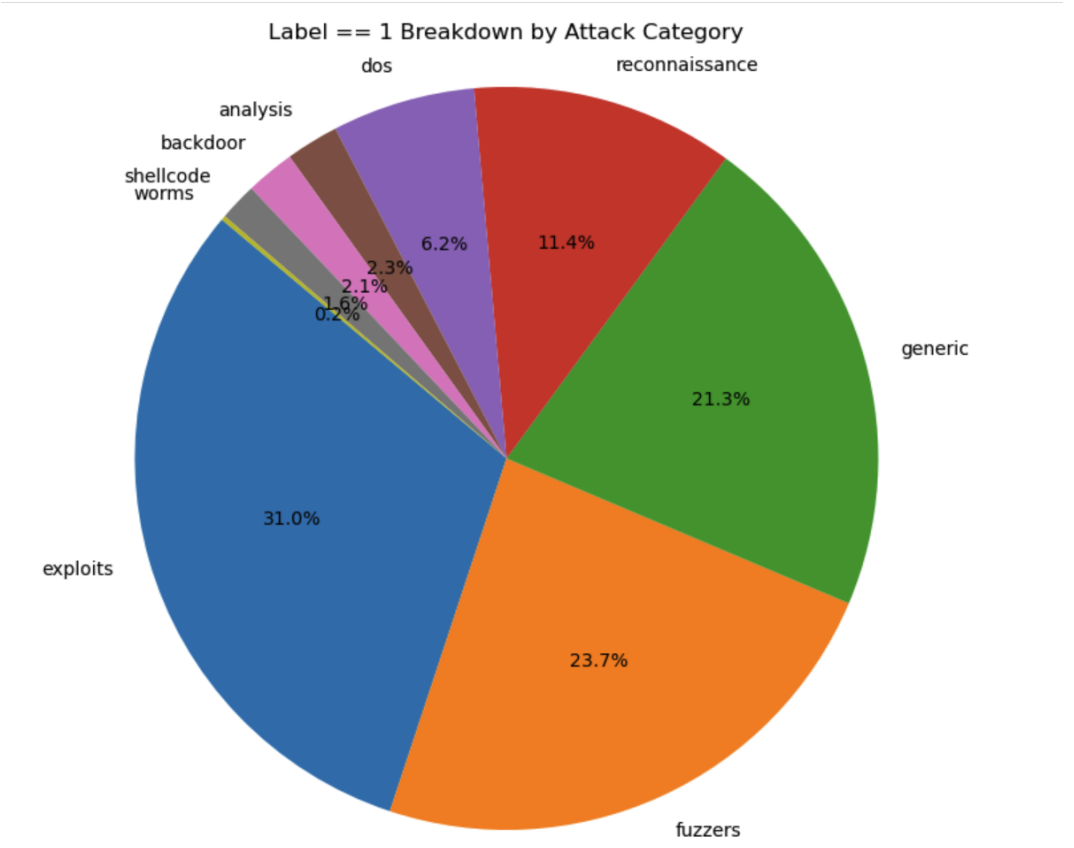


Figure 4: Pie Chart — Distribution of Attacks for Multiclass

✅ Best Parameters: {'max_depth': 20, 'min_samples_split': 5, 'n_estimators': 150}

📊 Classification Report (Multiclass – Attack Categories):

	precision	recall	f1-score	support
dos (#2)	0.469	0.283	0.353	1100
exploits (#3)	0.865	0.832	0.848	5487
fuzzers (#4)	0.941	0.892	0.916	4193
generic (#5)	0.978	0.857	0.914	3772
reconnaissance (#7)	0.830	0.786	0.808	2024
merged_rare (analysis, backdoor, shellcode, worms) (#99)	0.339	0.781	0.473	1109
accuracy			0.809	17685
macro avg	0.737	0.738	0.719	17685
weighted avg	0.846	0.809	0.819	17685

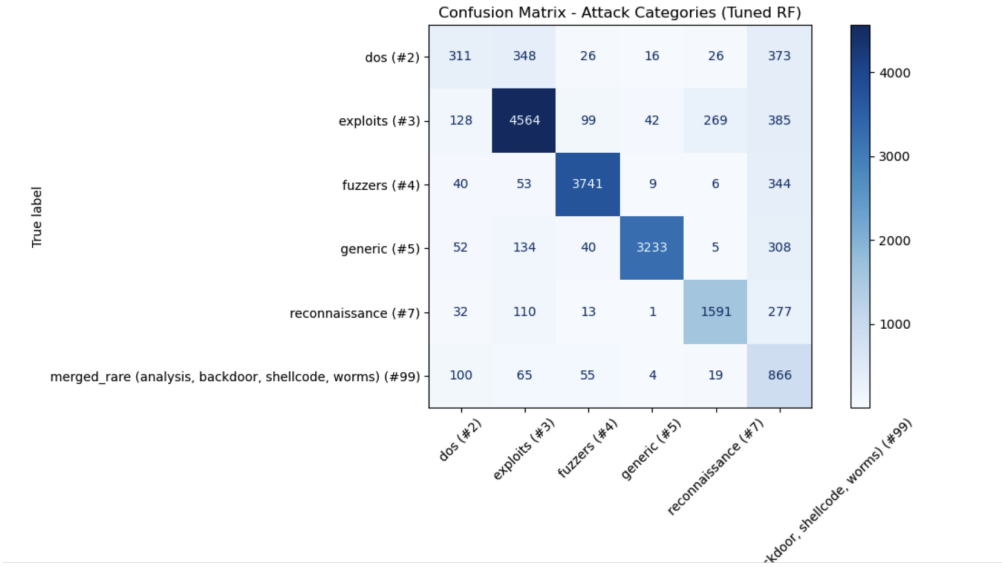


Figure 5: Confusion Matrix — Random Forest Multiclass Classifier

Appendix E: CatBoost

This appendix presents the full development process and evaluation results for the CatBoost model, which was applied for both binary classification (normal vs. attack) and multiclass classification (attack type). The process involved iterative tuning with Optuna, comparisons between default and optimized configurations, and the use of cost-sensitive learning strategies.

1. Binary Classification: Normal vs. Attack

The CatBoostClassifier was first used to predict whether a given log record represented a normal behavior (label = 0) or an attack (label = 1). Several configurations were explored:

Default Configuration:

- Iterations: 1000
- Depth: 8
- Learning Rate: 0.1
- Loss Function: Logloss
- Evaluation Metric: Accuracy
- GPU Acceleration: Enabled
- No Class Weights applied initially

Initial Results:

- Precision (1): 0.93
- Recall (1): 0.92
- F1-Score (1): 0.92
- Accuracy: 99%

Focused Tuning – Recall Optimization:

- Used Optuna to optimize recall
- Applied class weights to boost recall

- Result: Recall increased to 1.00; Precision dropped to 0.77

Balanced Tuning – Final Model (F1-Optimized):

- Used Optuna with F1-Score as the objective
- Result: Precision = 0.92, Recall = 0.92, F1-Score = 0.92, Accuracy = 99%

Table 17: CatBoost Binary Classification Report (Final)

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Normal (0)	1.00	1.00	1.00	387782
Attack (1)	0.92	0.92	0.92	17685
Accuracy		0.99		
Macro Avg	0.96	0.96	0.96	405467
Weighted Avg	0.99	0.99	0.99	405467

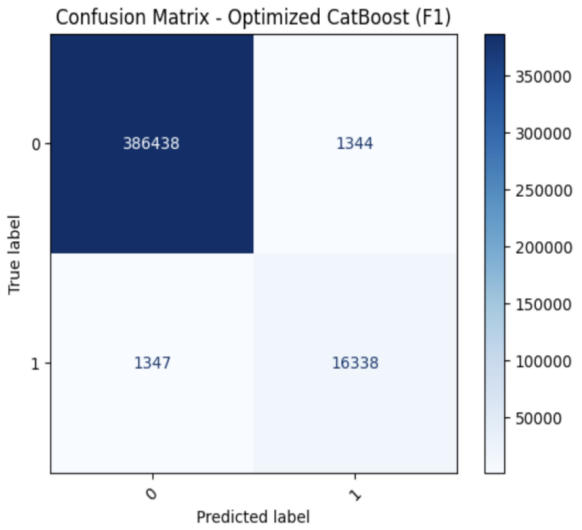


Figure 6: CatBoost Binary Classifier Confusion Matrix (Final)

2. Multiclass Classification: Attack Type

Initially, CatBoost was trained to classify 9 attack types. Due to severe imbalance, class weighting and tuning were introduced to improve minority class performance.

Steps and Strategies:

- **Step 1: Default Training (9 classes)** – Accuracy: 78%, F1 (macro): 54%
- **Step 2: Class Weighting** – Macro F1 improved to 62%
- **Step 3: Optuna Tuning** – Accuracy increased to 79%, F1 (macro): 64%
- **Step 4: Merged Classes (4 total)** – Accuracy jumped to 85%, F1 (macro): 85%
- **Step 5: Final Optuna (4 classes)** – Macro F1 improved to 86%

Table 18: CatBoost Multiclass Classification Report (Final 4-Class Model)

Class	**Precision**	**Recall**	**F1-score**	**Support**
Exploits (0)	0.84	0.86	0.85	8231
Fuzzers (1)	0.95	0.88	0.92	6290
Generic (2)	0.97	0.86	0.91	5658
Others (3)	0.71	0.82	0.76	6349
Accuracy		0.85		
Macro Avg	0.87	0.85	0.86	26528
Weighted Avg	0.86	0.85	0.86	26528

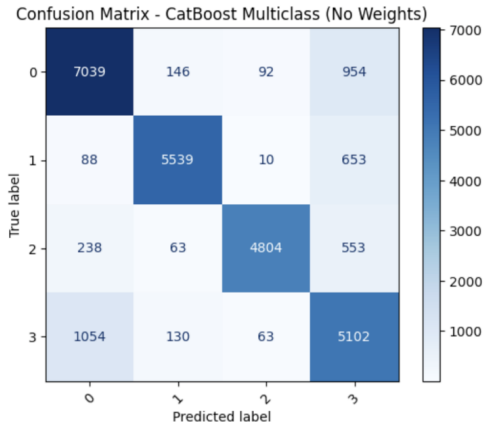


Figure 7: CatBoost Confusion Matrix Before assigning class weights

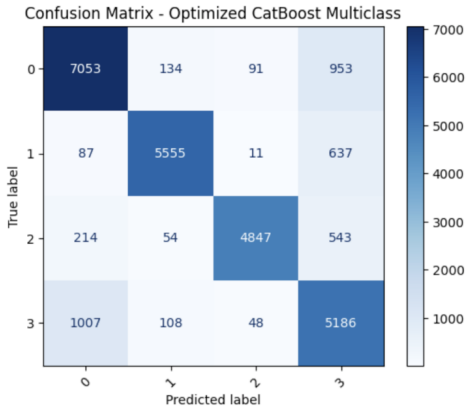


Figure 8: CatBoost Final Multiclass Confusion Matrix (4-Class Model)

Tuning Approach:

- **Optuna** was preferred over Grid Search due to its efficiency in large search spaces
- **Cost-Sensitive Modeling** was chosen instead of SMOTE to preserve the integrity of structured network data

Conclusion: CatBoost delivered some of the strongest overall results, with near-perfect binary classification and robust multiclass performance after class merging and hyperparameter tuning.

Appendix F: Deep Learning Pipeline

This appendix provides a comprehensive walkthrough of the development and optimization of the deep learning pipeline used for cyber threat detection. The pipeline was implemented in multiple stages, including early experiments with direct multiclass classification, extensive performance tuning, and finally a two-stage hierarchical architecture separating binary and multiclass prediction.

Initial Multiclass Attempts (Single Model)

We initially trained a direct multiclass classifier to distinguish between:

- **normal**
- **exploits**
- **fuzzers**
- **generic**
- **others** (merged from rare categories)

Model Architecture:

- 38 numerical features + 3 categorical (proto, state, service) embedded.
- 6 fully connected layers: $256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow$ output layer.
- Dropout rate: 0.2
- Loss: Focal Loss (= 2.0), class weighted
- Optimizer: Adam
- Epochs: 30
- Batch size: 512

Performance:

- **Accuracy:** 65%
- **Macro Avg F1-Score:** 0.32

- **Weighted Avg F1-Score:** 0.77
- *Problem: Extremely low precision for minority classes (e.g., 0.04 for exploits).*

Conclusion: Although recall improved due to Focal Loss, precision and class balance remained poor. As a result, we abandoned this approach in favor of a more modular and realistic pipeline.

Stage 1: Binary Classifier (Normal vs. Attack)

Architecture:

- Shared layers: 128 → 64 → 32 → 16
- Binary head: Dense → Sigmoid
- Loss: BCEWithLogitsLoss with pos_weight=5.0
- Optimizer: Adam
- Threshold Tuning: 0.4 to 0.6

Training Enhancements:

- Epochs: 80
- Learning rate decay: from 0.001 to 0.0005 at epoch 30
- Scheduler: ReduceLROnPlateau

Performance:

- **Attack F1-Score:** 89%
- **Attack Precision:** 88%
- **Attack Recall:** 90%
- **Accuracy:** 99%

Stage 2: Multiclass Attack Classification (Label == 1)

After successful binary filtering, only attack-labeled samples were passed into a specialized multiclass model.

Multiclass Head Architecture (Final Version):

- 6 hidden layers: 512 → 256 → 128 → 64 → 32 → 16
- Embeddings: proto (6), state (6), service (8)
- Dropout: 0.3
- Optimizer: AdamW
- Epochs: 500
- Loss: Focal Loss + Label Smoothing (smoothing=0.1, =1.5)
- Manual LR drops at 100, 200; scheduler applied

Result:

- **Accuracy:** 79%
- **Macro Avg F1:** 80%
- **Observation:** Performance on "others" class greatly improved with these changes.

Pipeline Overview

- Binary classifier trained on full dataset.
- Multiclass classifier trained only on attacks.
- Each head fine-tuned independently.

Final Pipeline Evaluation (Full Test Set)

Table 19: Final Classification Report — Two-Stage Deep Learning Pipeline

Class	**Precision**	**Recall**	**F1-score**	**Support**
Normal	1.00	0.99	0.99	387,782
Exploits	0.95	0.99	0.97	5,555
Fuzzers	0.49	1.00	0.66	4,279
Generic	1.00	1.00	1.00	3,164
Others	0.87	1.00	0.93	4,687
Accuracy	**0.99 (on 405,467 samples)**			
Macro Avg	0.86	1.00	0.91	405,467
Weighted Avg	0.99	0.99	0.99	405,467

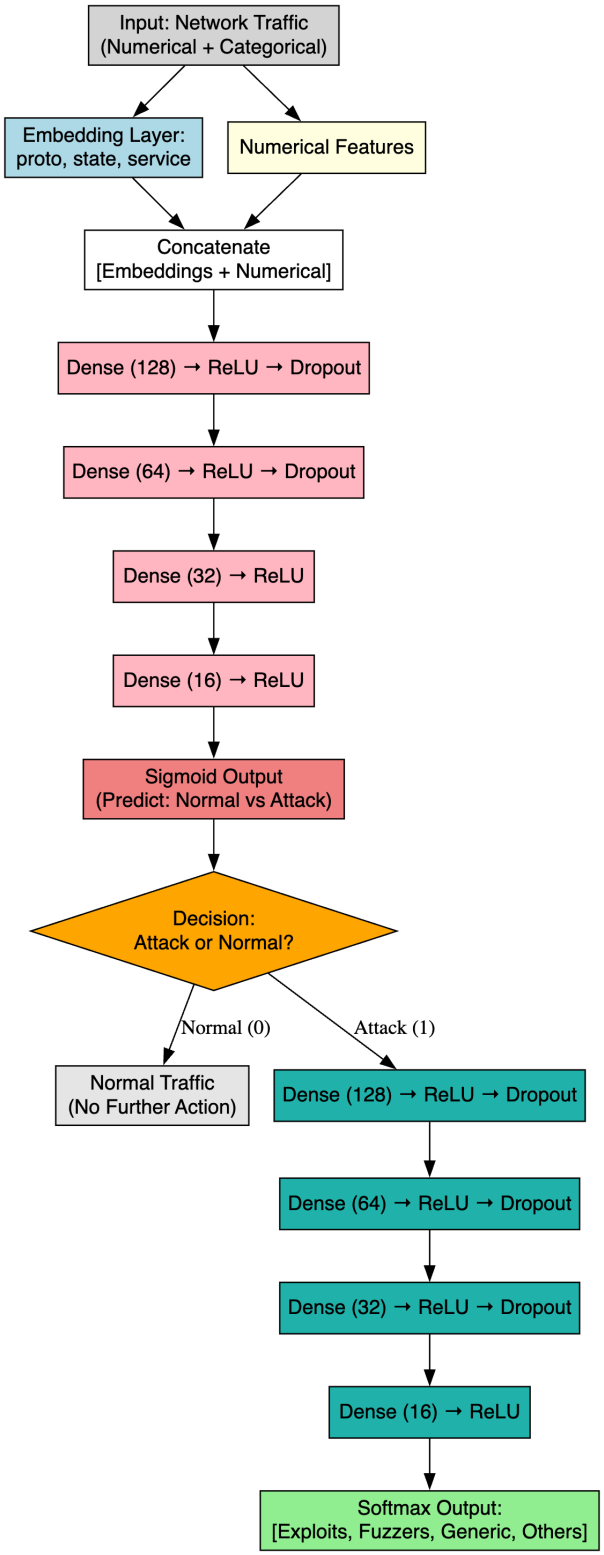


Figure 9: *Final Pipeline Architecture. Each sample passes through the binary model; if classified as attack, the multiclass head predicts the specific category.*

References

[1] Moustafa, Nour, and Jill 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). IEEE, 2015. `https://research.unsw.edu.au/projects/unsw-nb15-dataset`

[2] Prokhorenkova, Liudmila, et al. *CatBoost: unbiased boosting with categorical features*. Advances in Neural Information Processing Systems, 31, 2018. `https://catboost.ai`

[3] Lin, Tsung-Yi, et al. *Focal Loss for Dense Object Detection*. IEEE International Conference on Computer Vision (ICCV), 2017. `https://arxiv.org/abs/1708.02002`

[4] Akiba, Takuya, et al. *Optuna: A Next-generation Hyperparameter Optimization Framework*. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019.
`https://optuna.org`

[5] Pedregosa, F., et al. *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2011, pp. 2825–2830.
`https://scikit-learn.org`

[6] Paszke, A., et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Advances in Neural Information Processing Systems 32 (NeurIPS 2019).
`https://pytorch.org`