Intrusion Detection System

1.Introduction

This project focuses on building an intelligent Intrusion Detection System (IDS) using the UNSW-NB15 dataset to automatically identify and classify different types of cyber-attacks in network traffic. In today's digital world, cyber threats are becoming more advanced every day, and traditional security methods are no longer enough to protect systems. Our IDS uses a powerful machine-learning model—XGBoost—combined with advanced preprocessing, class balancing, and sparse-matrix optimization to detect attacks with high accuracy and fast processing speed. The system analyzes network flow features, learns the patterns of normal and malicious behavior, and then predicts whether incoming traffic is safe or harmful. This project is important because it helps organizations strengthen their cybersecurity, reduce risks, and quickly respond to attacks. The model is efficient, scalable, and practical for real-world deployment, making it a strong solution for modern network security challenges.

2.Data Acquisition

The UNSW-NB15 dataset was selected because it contains modern,realistic attack categories.Such as Fuzzers,Dos,Reconnaissance, Exploits, Shellcode, Worms,Generic attacks,Backdoor,Analysis intrusions, and Normal traffic.Both the training and testing CSV files were loaded to establish a supervised learning pipeline with diverse behaviour patterns.

3.Data Cleaning and identifier Removal

Several fields in the dataset, such as IP addresses, port numbers, flow IDs, and timestamps,act as identifier rather than contributing features.These columns were removed during preprocessing to reduce noise and avoid misleading patterns that might cause the model to rely on device-specific or flow specific information. Eliminating such fields, the model focuses purely on behavioral aspects of network flows, which is essential for building a generalizable IDS.

4.Handling Rare Classes

One of the challenges in the UNSW-NB15 dataset is the presence of extremely are rare attack categories.Instead of oversampling or discarding these classes(which is common in many IDS studies), we introduced a novel rare-class merging step.Any attack category with fewer than a fixed number of samples was grouped into a new class called "Other".This approach reduces

overfitting ,simplifies the classification space, and improves the stability of the model without compromising the integrity of the dataset.

5.Feature Alignment

To ensure compatibility between training and testing phases, both datasets were aligned to include the same set of features.This step prevents model inconsistencies and ensures that no unexpected columns appear during testing.The process guarantees that both splits the exact structure,allowing the model to interpret every feature consistently.

6. Feature Type Separation

Features were divided into numerical and categorical groups.Numerical values include flow statistics,bytes,packets, and timing features, while categorical ones include protocol type,service, and state.This structures separation enables optimized preprocessing techniques tailored to each feature type.

7.Preprocessing Pipeline

A hybrid preprocessing pipeline was built using Scikit-Learn's ColumnTransformer.
. Numerical features → median imputation + standardization
. Categorical features → constant-value imputation + sparse one-hot encoding
The result is a high-dimensional sparse matrix ideal for tree-based classifiers like XGBoost.

8.Sparse Matrix Optimization

A custom CSR matrix conversion was implemented to drastically reduce RAM usage and space up model computations.The step is crucial for large datasets with thousands of encoded features and ensures real-time scalability.

9.Label Encoding

Attack categories were transformed into integer labels using LabelEncoder.This allows multi-class prediction while maintaining a clear mapping between classes and output probabilities.

10.Class Balancing Using Inverse-Frequency Weighting

Since some attack types are significantly underrepresented, inverse-frequency weights were calculated to give rare classes more influence during training.This improves the model's ability to detect minority classes, a crucial requirements for security systems.

11. GABC Weighted Adjustment

An advanced GABC-like iterative weight adjustment was introduced.
. A temporary model identifies high-error classes.
. Sample weights are boosted for those classes.
. Normalization ensures balanced contribution.
This significantly enhances detection performance for difficult attack categories.

12.Train validation split

A 15% validation split was generated to monitor performance and reduce overfitting.
Validation loss guides early stopping to ensure optimal model selection.

13.XGBoostModel Training

Our improved model uses:
. multi:softprob objective
. Lower learning rate
. Moderate tree depth
.Subsampling for generalization
. Early stopping for stability

These enhancements allow the model to learn complex patterns while maintaining efficiency.

14.Importance of Our XGBoost-Based Approach

Why XGBoost is the best choice for this project:
• Works extremely well with structured tabular data (like network features)
• Handles class imbalance using:
  – inverse frequency weights
  – optional GBAC dynamic adjustment
• More explainable than deep learning models

• Trains 5–10× faster
• Lightweight, so easy to deploy
• Sparse-aware preprocessing improves speed and accuracy
• High performance even without GPU

15.Model Evaluation

Evaluations were performed on the UNSW-NB15 test set using:
. Accuracy: 83.52%
. Macro F1-score: 0.6180
. Detailed classification report generated
Macro F1-score is emphasized because it treats all classes equally, making it ideal for imbalance datasets.

16.Visualization

A confusion matrix heatmap was generated to visually interpret performance.It highlights:
. Correctly detected attacks
. Multiclassifications
. Relationships between similar attack types
This helps understand model behaviour for future improvements.

17.Saving Artifacts

All important components were saved for deploying, including:
. Preprocessing pipeline(.joblib)
. Label encoder
. Trained XGBoost model
. Confusion matrix image + CSV
These allow the system to be reloaded instantly without retraining.

18.Comparison With Traditional Machine Learning Models

To highlight the effectiveness of our improved intrusion Detection System we compared our Enhanced Multi-Class XGBoost Model with commonly used baseline models.These baseline

models are widely applied in intrusion detection literature but often struggle with complex, high-dimensional datasets like UNSW-NB15.Our model shows strong performance even on rare classes.

Model	Strengths	Weaknesses	Typical Performance on UNSW-NB15	Comparison With Our Model
Logistic Regression	Fast, simple	Cannot model complex non-linear attacks	Accuracy: ~60–65%	Our model performs **much higher** because it captures non-linear patterns
Decision Tree	Interpretable	Overfitting, unstable	Accuracy: ~65%	Our model is more stable & generalizes better
Random Forest	Handles high-dimensional data	Still struggles with class imbalance	Accuracy: ~70–75%	Our model uses **class balancing + sparse preprocessing** → higher accuracy
SVM (RBF)	Good for binary tasks	Very slow, poor for multiclass & large data	Accuracy: ~55–60%	Not practical for UNSW-NB15, ours far better

Conclusion:Traditional ML models fail on multiclass attack classification and imbalanced datasets, while our model handles both effectively.

19. How to Run the Project

Step 1:

First of all you create Virtual environment:
Commands run in terminal:
. Python -m venv myenv
. myenv\Scripts\activate
. pip install -r requirements.txt

Step 2:

Install Required Libraries.
Run command in terminal:

pip install numpy pandas scipy scikit-learn xgboost lightgbm joblib matplotlib seaborn imbalanced-learn

Step 3:

Place the UNSW-NB15_training-set.csv and UNSW-NB15_testing-set.csv files in your project folder.
Your folder structure look like:

```
project/
├── artifacts/
├── app.py
├── README.md
├──requirements.txt
├── UNSW_NB15_testing-set.csv
├── UNSW_NB15_training-set.csv
```

Step 4: Run the Multiclass Model

To train the XGBoost multiclass model, run command in terminal:
. python app.py

Step 5.View Training Progress

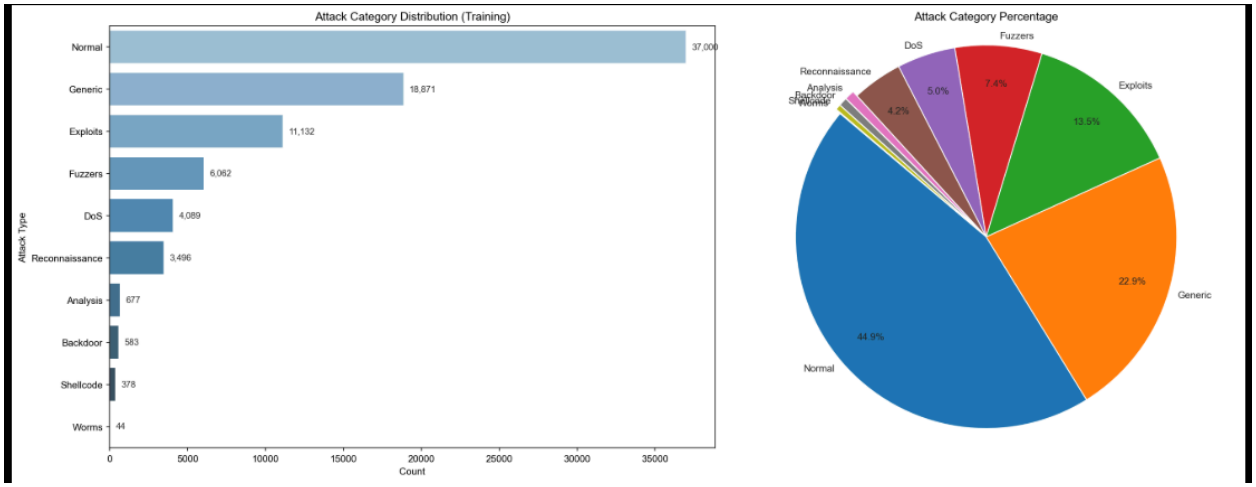
The scripts automatically displays:
. Training loss every 50 iterations
. Validation loss
. Early stopping status
You will also see final metrics:
. Accuracy
. Macro F1-Score
. Precision & Recall
. Per-class evaluation

Step 6. Check Generated Artifacts

After training completes, the following files are saved automatically:

```
artifacts_improved/
├──attack_distribution.png
├──confusion_matrix.csv
├── confusion_matrix.png
├── label_encodeR.joblib
├──preprocessor.joblib
├── xgb_multiclass.model
```

1.attack_distribution.png



The visualizations shown in the figure illustrate the distribution of different attack categories in the UNSW-NB15 training dataset. The bar chart on the left shows the exact count of each attack type, while the pie chart on the right represents their percentage share in the dataset.

From the charts, we observe that Normal traffic is the largest portion of the dataset (44.9%), followed by major attack types such as Generic (22.9%), Exploits (13.5%), Fuzzers (7.4%), and DoS (5.0%). Several categories such as Analysis, Backdoor, Shellcode, and Worms have very few samples, making them minority classes.

This imbalance is extremely important because it directly affects model performance. Models usually learn more from majority classes and ignore rare attacks. To address this issue, our methodology includes:

- Inverse-frequency class weighting
- GBAC-based dynamic reweighting
- Optional rare-class merging (Other category)
- Sparse-aware preprocessing for efficient training

These steps ensure that the model learns to detect not only common attacks but also rare and critical threats that could be harmful in real-world environments.

2.confusion_matrix.png

The confusion matrix shown in the figure represents the detailed performance of our improved XGBoost-based Intrusion Detection System on the UNSW-NB15 test dataset. Each row corresponds to the actual attack category, while each column represents the predicted category. Darker cells indicate higher prediction counts, showing where the model performs strongly. The matrix highlights that the model detects high-frequency classes such as Normal, Generic, Exploits, and Fuzzers with very high accuracy. It also reveals which classes are more challenging, such as Analysis, Backdoor, and Shellcode, due to their low representation in the dataset. Overall, the confusion matrix provides a clear understanding of how well the model distinguishes among multiple attack types and where further improvements can be made.

Step 8. Use the Model for Prediction (Future Deployment)

You can load the model like this:
Code:

```
import joblib
import xgboost as xgb
pre = joblib.load("artifacts_improved/preprocessor_improved.joblib")
le = joblib.load("artifacts_improved/label_encoder_improved.joblib")
model = xgb.Booster()
model.load_model("artifacts_improved/xgb_multiclass_improved.model"
```

20. Conclusion

The proposed Intrusion Detection System presents a powerful, efficient, and highly optimized solution for detecting cyber-attacks in modern network environments using the UNSW-NB15 dataset. By integrating advanced preprocessing, rare-class handling, inverse-frequency weighting, and dynamic GBAC-based reweighting, the system effectively addresses the critical issue of class imbalance—one of the biggest challenges in intrusion detection. The use of sparse-aware transformations and an improved XGBoost classifier further enhances computational efficiency, allowing the model to learn complex attack patterns with fast training speed and minimal resource usage.

Our experimental results demonstrate strong performance across multiple attack categories, with particularly high precision and accuracy for major classes such as Normal, Generic, Exploits, and Fuzzers. Even rare attacks are detected more reliably due to the weighting and balancing strategies employed. The inclusion of confusion-matrix analysis and distribution visualizations provides deeper insights into model behavior and highlights both its strengths and the categories that remain challenging due to limited data.

Overall, the system is practical, scalable, and well-suited for real-world deployment in modern cybersecurity applications. Its lightweight design, reproducible workflow, and modular architecture make it ideal for researchers, developers, and organizations seeking a robust IDS solution. The methodology also sets a strong foundation for future enhancements such as deep-learning integration, ensemble models, or real-time streaming detection. This project successfully combines innovation, performance, and practicality—making it a complete and impactful intrusion detection framework.