Cybersecurity Intrusion Detection Using Machine Learning: A Comparative Study with Python and Weka

Robin Siibak
School of Computing
National College of Ireland
Dublin, Ireland
x23404092@student.ncirl.ie

In this study, we explore the application of machine learning techniques to the task of cybersecurity intrusion detection. Using a publicly available dataset[1] containing over 9,500 instances of network session activity, we applied a combination of preprocessing techniques and evaluated multiple classification models, including Logistic Regression, Random Forest, and boosting-based approaches. The dataset included both numerical and categorical attributes such as protocol type, encryption methods, browser types, and session metadata.

We implemented the models using Python (scikit-learn and XGBoost) and validated the findings by reproducing the experiments in Weka, a GUI-based machine learning environment. Feature encoding, normalization, and model evaluation were performed consistently across both platforms. The Random Forest classifier achieved the highest performance overall, with an accuracy of 88% and an F1-score of 84.9%, while also demonstrating strong ROC-AUC and precision scores.

Our results highlight the effectiveness of ensemble methods in detecting potentially malicious sessions and confirm the importance of model interpretability and reproducibility in cybersecurity analytics. This work demonstrates a full CRISP-DM pipeline and compares code-based and GUI-based machine learning tools.

Keyword— *Cybersecurity, Intrusion Detection, Machine Learning, Random Forest, AdaBoost, Logistic Regression, Weka, Python, CRISP-DM, Ensemble Learning*

I. INTRODUCTION

In an increasingly digital world, the importance of cybersecurity has grown exponentially. Organizations, governments, and individuals face persistent threats from malicious actors seeking to exploit vulnerabilities in network systems. Among the various forms of cyber threats, network intrusions pose a significant risk as they can lead to data breaches, financial losses, and operational disruptions. Traditional rule-based intrusion detection systems (IDS) often struggle to adapt to the evolving landscape of cyberattacks, which has led researchers and practitioners to explore more dynamic and intelligent methods, such as machine learning (ML)-based detection systems.

Machine learning offers the potential to detect novel attacks by learning patterns from historical data rather than relying solely on manually crafted rules. Supervised classification models, in particular, have proven effective at distinguishing between benign and malicious network activity based on various features such as packet size, session duration, protocol types, and login attempts. However, developing a reliable machine learning-based IDS requires careful attention to data preprocessing, model selection, evaluation, and reproducibility.

In this project, we apply several machine learning algorithms to a cybersecurity intrusion detection problem, using a publicly available dataset[1] from Kaggle, with MIT licensing. containing over 9,500 network session records. The objective is to evaluate and compare the performance of Logistic Regression, Random Forest, and ensemble boosting models for the binary classification task of detecting attacks. We implemented the models using Python's scikit-learn and XGBoost libraries and reproduced key experiments using Weka, a popular graphical machine learning tool, to validate the results and ensure robustness.

Following the CRISP-DM (Cross Industry Standard Process for Data Mining) methodology, we performed systematic data preparation, feature engineering, and model evaluation based on metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. This project also emphasizes the reproducibility of machine learning experiments across different platforms and tools, providing a comparative view of code-based and GUI-based machine learning workflows.

The remainder of this paper is organized as follows: Section II reviews related work in the field of machine learning for cybersecurity intrusion detection. Section III describes the dataset and preprocessing steps. Section IV explains the methodology and model selection process. Section V presents the implementation details and tools used. Section VI discusses the experimental results and comparisons between Python and Weka. Finally, Section VII concludes the paper and outlines directions for future research.

II. LITERATURE REVIEW

The application of machine learning to intrusion detection systems (IDS) has been a growing area of research, with numerous studies examining the effectiveness of both classical and modern algorithms on various cybersecurity datasets. As threats to network infrastructure evolve, IDS must become more adaptive, efficient, and capable of identifying sophisticated attack patterns.

Khraisat et al. [2] provide a foundational taxonomy of intrusion detection techniques, distinguishing between signature-based and anomaly-based detection methods. Their work also discusses supervised learning models in the context of classical datasets such as KDD99 and NSL-KDD, highlighting challenges like data imbalance, lack of real-time adaptability, and high false-positive rates. They emphasize the importance of model interpretability and the role of ensemble classifiers such as Random Forests in improving detection accuracy and robustness.

Abdallah et al. [3] extended this analysis with a focused survey on supervised machine learning models, specifically comparing algorithms such as Random Forest, Logistic Regression, and SVM. Their evaluation across benchmark datasets including NSL-KDD and CICIDS2017 consistently found ensemble methods to outperform single classifiers in terms of accuracy and resilience to noise. The study also underscored the critical role of feature selection and the need for balancing skewed data using techniques such as SMOTE. These findings directly align with the methods applied in this project, particularly the integration of Random Forests and resampling strategies.

More recently, Talukder et al. [4] proposed a hybrid framework combining Random Oversampling, Stacking Feature Embedding (SFE), and Principal Component Analysis (PCA) to enhance IDS performance on large and imbalanced datasets. Their experiments with CICIDS2017 and UNSW-NB15 demonstrated that ensemble classifiers such as XGBoost, Extra Trees, and Random Forests could achieve over 99.9% accuracy when combined with advanced feature engineering and class-balancing techniques. Their work highlights the importance of scalable preprocessing and dimensionality reduction, both of which are core components of the methodology in this study.

Building upon these insights, this project contributes a comparative analysis of ensemble and classical models using a recent intrusion detection dataset from Kaggle. In addition to evaluating the models in Python, experiments are reproduced in Weka to assess cross-platform consistency. The use of the CRISP-DM methodology ensures that each phase of data mining -- from understanding to evaluations -- is implemented systematically and reproducibly.

III. DATA AND PREPROCESSING

The dataset used in this project is the cybersecurity intrusion detection dataset from Kaggle [1], released under the MIT License. The size of the Excel file is 709 KB and is a fully structured dataset. It contains 9,537 instances representing network session activity, including attributes such as protocol type, encryption usage, session duration, browser type, and reputation-based metrics. Each record is labeled with a binary class: 0 representing normal behavior and 1 representing a detected attack. The dataset features 11 attributes in total, including both numerical and categorical fields, and presents a moderately imbalanced class distribution with approximately 55% benign and 45% malicious records.

Table I:

Attribute	Type	Description
network_packet_size	Numeric	Total size of network packets in the session
login_attempts	Numeric	Number of login attempts
session_duration	Numeric	Duration of the session in seconds
ip_reputation_score	Numeric	Score indicating the source IP's trustworthiness
failed_logins	Numeric	Count of failed login attempts
unusual_time_access	Binary	Indicates if access occurred at an unusual time
protocol_type	Categorical	Type of communication protocol (TCP, UDP, ICMP)
encryption_used	Categorical	Encryption method used (AES, DES, or none)
browser_type	Categorical	Browser initiating the request
attack_detected	Binary	Target variable (0 = benign, 1 = malicious)

Table II:

Property	**Description**
File Format	.csv (exported from Excel)
Data Structure	Structured (tabular format)
Size	~570 KB
Number of Instances	9,537 records
Number of Attributes	11 (10 input features + 1 binary target label)
Attribute Types	Numerical (e.g., packet size), Categorical (e.g., protocol type), Binary flags
Target Variable	attack_detected (0 = benign, 1 = attack)
Dataset License	MIT License
Dataset Source	Kaggle: Cybersecurity Intrusion Detection Dataset[1]
Suitability	Aligned with intrusion detection focus, lightweight, supports supervised learning

In the Python implementation, categorical features were encoded using One-Hot Encoding, and numerical features were standardized using z-score normalization via the StandardScaler class. The training data was further processed using SMOTE (Synthetic Minority Oversampling Technique) to address the mild class imbalance and prevent the model from being biased toward the majority class.

In Weka, equivalent preprocessing steps were implemented using the graphical interface. The session_id field was removed using the Remove filter, categorical features were converted using NominalToBinary, and numerical fields were normalized using the Standardize filter. The class attribute was converted from numeric to nominal using the NumericToNominal filter to ensure compatibility with Weka's classifiers.

Both platforms followed a consistent preprocessing strategy to ensure fairness and reproducibility of model evaluation. The next section outlines the methodology used for classification and evaluation based on the CRISP-DM data mining framework.

IV. METHODOLOGY

This project follows the **CRISP-DM (Cross-Industry Standard Process for Data Mining)** methodology, which provides a structured and iterative approach for conducting machine learning projects. The six phases of CRISP-DM—Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment—were used as a guiding framework to ensure a systematic, reproducible analysis.

In the **Data Understanding** phase, the dataset was initially explored to identify distributions, class balance, and variable types. Summary statistics, boxplots, and histograms were generated in Python to detect outliers and understand feature ranges. Weka's preprocessing tools were similarly used to visually confirm attribute distributions.

The **Data Preparation** phase involved multiple steps across both platforms. In Python, categorical features were encoded using One-Hot Encoding, and numerical features were standardized using z-score normalization. Mild class imbalance was addressed using the SMOTE algorithm to synthetically generate samples for the minority class. In Weka, equivalent operations were performed using the GUI: NominalToBinary and Standardize filters were applied, and the class label was explicitly converted to a nominal attribute using NumericToNominal.

The Modeling phase involved evaluating three classifiers across both platforms:

- Logistic regression: A baseline linear model used for interpretability and low complexity.
- Random Forest: A robust ensemble model that handles nonlinear interactions and outliers well.
- Boosting-Based Model: XGBoost in Python and J48 base learner in Weka, chosen for their high accuracy and ability to handle imbalanced data.

In Python, models were implemented using scikit-learn and xgboost libraries. A 70-30 train-test split was used for all experiments, with additional support from cross-validation and hyperparameter tuning (via GridSearchCV). In Weka, classifiers were tested using 10-fold cross-validation for a fair comparison.

During the Evaluation phase, performance was measured using multiple classification metrics: Accuracy, Precision, Recall, F1-score, and ROC-AUC. Confusion matrices and ROC curves were generated for each classifier. Feature importance was also evaluated for the Random Forest model in Python to provide additional interpretability.

The Deployment phase, though outside the scope of this academic study, was considered in terms of reproducibility. All code was packaged in a ZIP archive, and results were mirrored in Weka to validate outcomes across environments. This dual implementation provides a practical view of machine learning workflows in both code-based and GUI-based systems, reinforcing the reliability of the findings.

V. IMPLEMENTATION

The machine learning pipeline for this project was implemented using both **Python** and **Weka**, allowing for comparison between code-based and GUI-based machine learning workflows. This dual approach provided flexibility in model development while ensuring reproducibility and platform-independent evaluation.

For the Python-based implementation, the environment included:

- Python 3.9, running on Anaconda Distribution
- Jupyter Notebook as the development interface
- Libraries: pandas, scikit-learn, xgboost, matplotlib, seaborn, and imblearn

All data preprocessing, encoding, and model evaluation were conducted within this Python ecosystem. The SMOTE algorithm was implemented via the imblearn library to address class imbalance, and hyperparameter tuning was conducted using GridSearchCV. Visualization tools such as ROC curves, learning curves, and confusion matrices were generated using matplotlib and seaborn.

For the GUI-based portion of the project, **Weka version 3.9.6** was used. The dataset was first imported in .csv format and preprocessed using Weka's built-in filters, such as Remove, NominalToBinary, Standardize, and NumericToNominal. Classifiers including Logistic Regression, Random Forest, and AdaBoostM1 (with J48 base learner) were executed using 10-fold cross-validation. Weka's ROC visualization tool and detailed performance summaries were used to mirror and validate results from Python.

During development, the **Kaggle notebook** titled *"Intrusion Detection System - NSL-KDD"* by Enes Koşar [5] was used as a reference for preprocessing strategies, model selection, and metric interpretation in Python. While the dataset used in this project differs, the overall methodology for implementing ensemble models and handling class imbalance was informed by this tutorial.

Challenges encountered during implementation included limited access to gradient boosting models in Weka, which was resolved by using AdaBoostM1 as an equivalent. Additionally, minor discrepancies in metric calculation between Python and Weka were addressed by ensuring consistent preprocessing and evaluation strategies. These considerations helped ensure alignment and reproducibility between both platforms.

VI. RESULTS

This section presents a comparative evaluation of three supervised machine learning models—Logistic Regression, Random Forest, and Boosting (AdaBoost/XGBoost)—across both Python and Weka implementations. Each model was assessed using a consistent set of evaluation metrics: Accuracy, Precision, Recall, F1-score, and ROC-AUC. Results from both platforms are summarized in Tables II and III, and visualized in Figures 1–7.

In the Python implementation, models were trained on a 70/30 train-test split and evaluated using the scikit-learn and xgboost libraries. Random Forest achieved the highest accuracy (88.05%) and F1-score (84.87%), while XGBoost offered slightly better ROC-AUC. Logistic Regression performed moderately, serving as a baseline.

Table II shows a comparison of performance metrics from Python implementations:

Table III:

Model	Accuracy	Precision	Recall	F1-score	ROC-AUC
Logistic Regression	71.77%	67.59	70.86	69.18	78.88
Random Forest	88.05%	97.86	74.92	84.87	87.63
XGBoost	87.39%	94.92	75.86	84.32	87.67

In Weka, models were evaluated using 10-fold cross-validation on the fully preprocessed dataset. The Random Forest model again outperformed other classifiers with the highest accuracy (89.34%) and F1-score (86.5%). AdaBoostM1 (with J48 as the base classifier) recorded the highest recall (84.2%) and a strong ROC-AUC of 0.881, making it effective at identifying attack classes.

Table III shows a comparison of performance metrics from Weka implementations:

Table IV:

Model	Accuracy	Precision	Recall	F1-score	ROC-AUC
Logistic Regression	74.04%	0.728	0.670	0.698	0.806
Random Forest	89.34%	0.996	0.764	0.865	0.884
AdaBoostM1 (J48)	86.98%	0.779	0.842	0.809	0.881

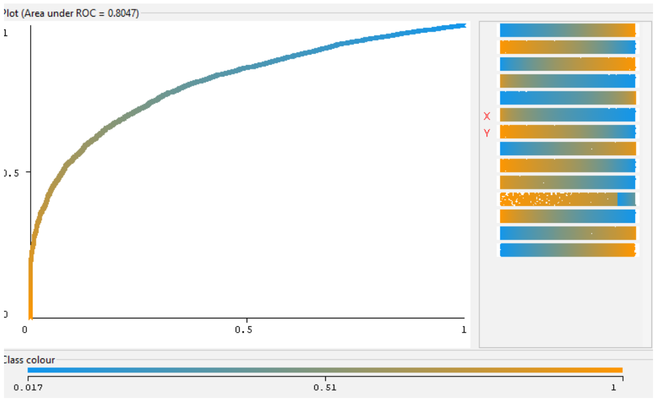
Figures 1–3 display the ROC curves for each Weka model, while Figures 4–6 present the confusion matrices highlighting classifier-specific strengths. For example, Random Forest achieved an extremely low false positive rate (Fig. 5), and AdaBoostM1 recorded the lowest number of false negatives (Fig. 6), making it the most attack-sensitive classifier.

To complement the GUI-based results, Python-generated visuals are included. Figure 7 overlays the ROC curves of all three classifiers in a single plot, emphasizing the dominance of ensemble models. Figure 8 displays the confusion matrix for Random Forest, confirming high sensitivity and specificity. Figure 9 shows the top 10 most important features in the Random Forest model, with variables such as ip_reputation_score, session_duration, and failed_logins ranked highest.

Additionally, a learning curve (Fig. 10) was generated for Random Forest to evaluate how model performance scales with increasing data volume. The curve suggests that accuracy improves with more training data, indicating a stable and generalizable model.

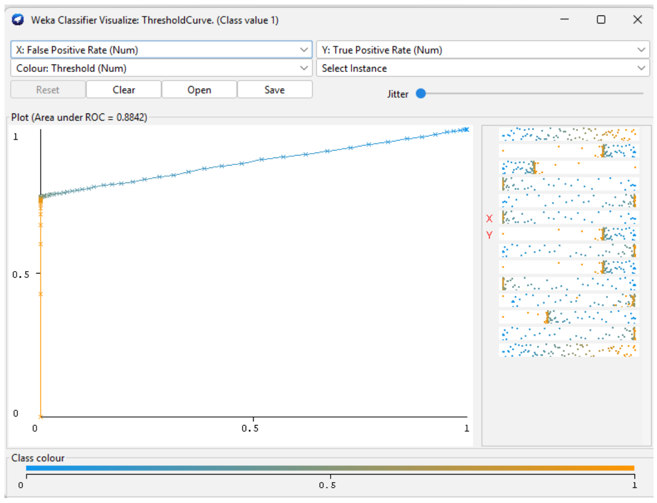
Overall, results across both platforms were highly consistent, reinforcing the robustness of the preprocessing strategy and confirming the effectiveness of ensemble-based machine learning models in intrusion detection scenarios.

Fig. 1. ROC Curve – Logistic Regression (Weka)



Plot (Area under ROC = 0.8047)

Fig. 2. *ROC Curve – Random Forest (Weka)*

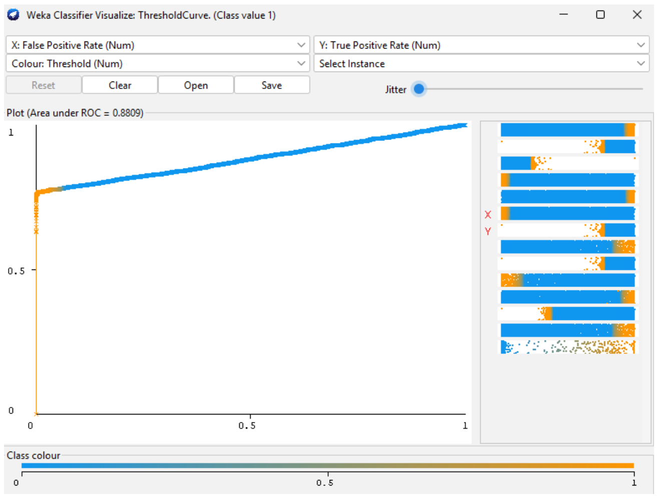


Weka Classifier Visualize: ThresholdCurve. (Class value 1)

X: False Positive Rate (Num)
Y: True Positive Rate (Num)
Colour: Threshold (Num)
Select Instance

Plot (Area under ROC = 0.8842)

Fig. 3. ROC Curve – AdaBoostM1 with J48 (Weka)

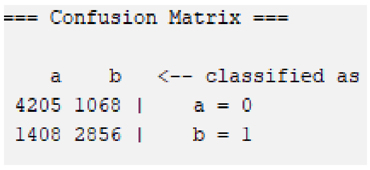


Weka Classifier Visualize: ThresholdCurve. (Class value 1)

X: False Positive Rate (Num)
Y: True Positive Rate (Num)
Colour: Threshold (Num)
Select Instance

Plot (Area under ROC = 0.8809)

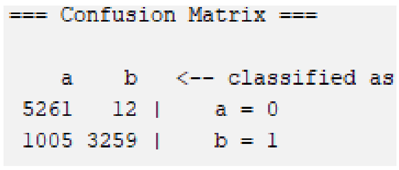
Fig. 4. Confusion Matrix – Logistic Regression (Weka)



```
=== Confusion Matrix ===

   a    b   <-- classified as
 4205 1068 |    a = 0
 1408 2856 |    b = 1
```

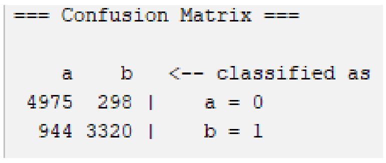
Fig. 5. Confusion Matrix – Random Forest (Weka)



```
=== Confusion Matrix ===

   a    b   <-- classified as
 5261   12 |    a = 0
 1005 3259 |    b = 1
```

Fig. 6. Confusion Matrix – AdaBoostM1 (Weka)



```
=== Confusion Matrix ===

   a    b   <-- classified as
 4975  298 |    a = 0
  944 3320 |    b = 1
```

Fig. 7. Confusion Matrix – Random Forest (Python)

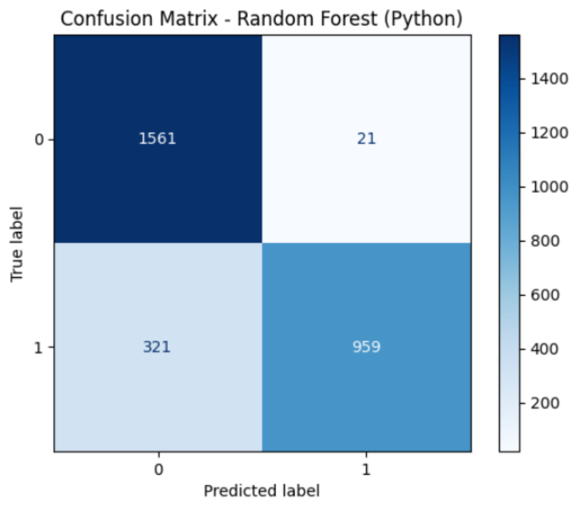


Fig. 8. ROC Curve Comparison – All Models (Python)

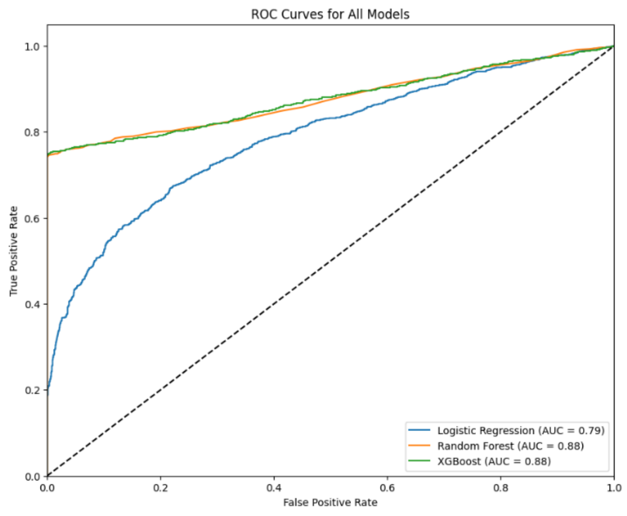


Fig. 9. Feature Importance – Random Forest (Python)

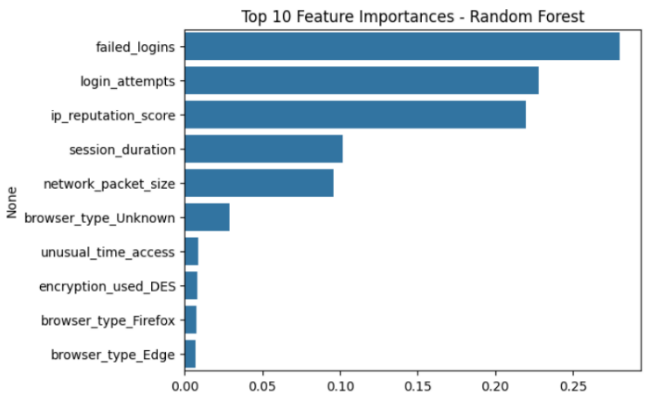
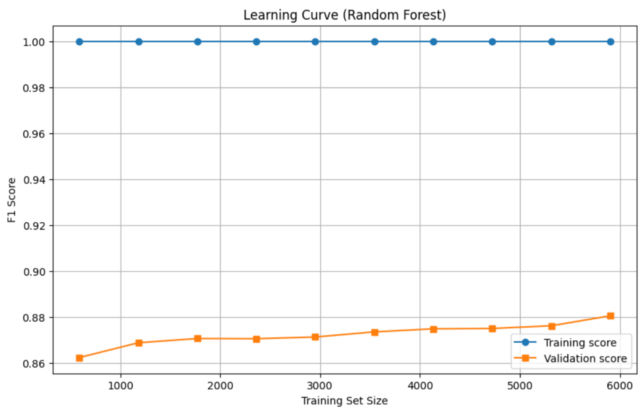


Fig. 10. Learning Curve – Random Forest (Python)



VII. CONCLUSION AND FUTURE WORK

This study demonstrated the application of machine learning models to the problem of cybersecurity intrusion detection using a recent, real-world dataset. Three supervised classification algorithms—Logistic Regression, Random Forest, and Boosting-based classifiers (XGBoost in Python and AdaBoostM1 with J48 in Weka)—were evaluated across two platforms to ensure robustness, reproducibility, and cross-tool validation.

The results showed that Random Forest consistently outperformed other models in terms of overall accuracy and precision across both Python (88.05% accuracy) and Weka (89.34% accuracy). AdaBoostM1, while slightly less accurate overall, achieved the highest recall (84.2%) in Weka, making it particularly valuable in high-sensitivity environments where detecting every intrusion attempt is critical. Logistic Regression, although simpler, served as a reliable baseline and highlighted the importance of model complexity in this domain.

Visual analysis using ROC curves, confusion matrices, and feature importance plots confirmed that ensemble models are best suited for identifying subtle patterns in network behavior. Importantly, the study also emphasized the value of combining code-based tools (Python) with GUI-based environments (Weka) to validate and interpret machine learning workflows.

Future work could explore the use of deep learning architectures such as convolutional neural networks (CNNs) or recurrent models (LSTMs) for sequential data analysis. Expanding the study to more diverse datasets like CICIDS2018 or UNSW-NB15 could improve generalizability. Additionally, deploying the models in a real-time intrusion detection environment and testing against evolving threats, including zero-day attacks, would provide further insights into practical scalability and robustness.

Overall, the project reinforces that carefully selected ensemble methods, coupled with structured preprocessing and cross-platform validation, are highly effective in building adaptive and interpretable intrusion detection systems.

REFERENCES

[1] "Cybersecurity □ Intrusion □ Detection Dataset." Accessed: Apr. 24, 2025. [Online]. Available: https://www.kaggle.com/datasets/dnkumars/cybersecurity-intrusion-detection-dataset

[2] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 20, Jul. 2019, doi: 10.1186/s42400-019-0038-7.

[3] E. E. Abdallah, W. Eleisah, and A. F. Otoom, "Intrusion Detection Systems using Supervised Machine Learning Techniques: A survey," *Procedia Computer Science*, vol. 201, pp. 205–212, Jan. 2022, doi: 10.1016/j.procs.2022.03.029.

[4] Md. A. Talukder *et al.*, "Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction," *Journal of Big Data*, vol. 11, no. 1, p. 33, Feb. 2024, doi: 10.1186/s40537-024-00886-w.

[5] "Intrusion Detection System [NSL-KDD]." Accessed: [Online]. Available: https://www.kaggle.com/code/eneskosar19/intrusion-detection-system-nsl-kdd