National College of Ireland

Enhancing Cloud Security with Machine Learning-Based Intrusion Detection Systems

MSc Research Project
Cloud Computing

Sai Bhanu Prasad Pacha
Student ID: x23213167

School of Computing
National College of Ireland

Supervisor: Yasantha Samarawickrama

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Sai Bhanu Prasad Pacha

Student ID: x23213167

Programme: MSc in Cloud Computing **Year:** 2025-2026

Module: MSc Research Project

Supervisor: Yasantha Samarawickrama

Submission Due Date: 28/01/2026

Project Title: Enhancing Cloud Security with Machine Learning Based Intrusion Detection Systems

Word Count: 8984 **Page Count:** 22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Sai Bhanu Prasad Pacha

Date: 28/01/2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□

Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Enhancing Cloud Security with Machine Learning-Based Intrusion Detection Systems

Sai Bhanu Prasad Pacha
X23213167

Abstract

Network intrusion detection systems have severe problems such as high false positive rates, imbalance of classes in training data, and disparity in the laboratory performance and production deployment needs. This study builds and analyses an ensemble machine learning model by integrating the Random Forest and XGBoost model to perform binary classification of network flows based on the CSE-CIC-IDS-2018 dataset. It preprocesses 723,337 samples and standardizes 69 network flow feature using network flow using preprocessing and stratified train-validation-test splitting. The accuracy of individual models is 87.7% (Random Forest) and 87.6% (XGBoost), whereas the weighted ensemble exhibits a high accuracy of 87.84% and a high precision and recall value of 95.2% and 78.3% respectively. The system runs on AWS EC2 through Docker containerization, where the median inference latency of the Flask REST API is 376ms and median server-side processing time is 42ms. Extensive monitoring solutions based on Prometheus and Grafana help track the performance in real-time, and a set of custom analytics solutions allow analysis of the latency distribution, evaluate its accuracy through confidence, and suggest the optimal threshold. The deployment has been shown to be 100% available during hours of testing with steady performance characteristics, indicating that it is ready to be deployed to production to provide real-time network intrusion detection with balanced accuracy and performance.

1 Introduction

1.1 Background and Motivation

Digital transformation of computing infrastructure has also presented major cybersecurity challenges (Saeed et al., 2023). Network intrusion attacks have become more sophisticated and larger in size, and threat actors have adopted sophisticated methods in order to breach systems, steal sensitive information, and disrupt vital services. A recent report has indicated that the current cybercrime will cost the world 10.5 trillion USD each year by 2025, which explains the need to have well-comprising security systems (Pandey and Kapoor, 2025).

Conventional signature based intrusion detection system (IDS) though efficient with known attacks has inherent detection challenges against new and zero-day attacks. These systems are based on the predefined patterns and rules, thus, being reactive, not proactive (Agaromoorthy et al., 2023). Since attackers are constantly devising new methods of exploitations, the difference between the evolution of threats and detection abilities increases. Machine learning (ML) techniques have become one of the possible solutions to this problem, providing the capability of learning intricate patterns based on network traffic data and identifying known and unidentified intrusions (Halbouni et al., 2022).

Ensemble learning techniques, utilizing multiple models of ML to enhance prediction accuracy and robustness have been shown to perform even better in many fields such as cybersecurity (Alabdulatif, 2025). Using their complementary capabilities, ensemble methods are able to attain increased detection rates and reduced false positive rates, a key attribute to be able to deploy an algorithm in production. The combination of ML-based IDS with cloud infrastructure and real-time monitoring systems can be seen as a great step toward practical and scalable network security solutions.

1.2 Problem Statement

Although substantial research advancement has been made on the field of ML based intrusion detection, there are a number of major challenges that prevent the implementation of such systems within the production setup. To begin with, the current IDS systems tend to have high false positives, and they produce large amounts of alarms due to innocent network traffic and overload security experts with alarm fatigue (Neupane et al., 2022). This decreases the efficiency of operation and can cause critical threats to be missed in among false alarms.
Second, the natural class distribution of network traffic datasets: In benign flows, while there are a very small number of attacks, has proven to be challenging to model training and evaluation (Milosevic and Ciric, 2022). Most ML models are biased towards the majority class so that they do not detect minority attack classes despite having high overall model accuracies. The problem with this limitation is especially problematic with rare but critical types of attacks. Third, many research articles state the high accuracy on benchmark datasets, but hardly any of them concern the practical deployment needs such as the inference latency in real-time, scalability, and compatibility with existing monitoring infrastructure. The issue of performance in the lab and production is another major limitation to the large-scale use of ML-based IDS solutions. There is a distinct demand of overall systems which do not only guarantee high detection rates but also prove operational feasibility by implementing through the cloud, tracking their performance and provisioning of actionable analytics.

1.3 Research Question

The thesis will answer the following research question:
What does it take ensemble machine learning models to achieve high-precision network intrusion detection with low false positive rates and real-time inference on a cloud production environment?
In particular, this study explores: (1) the performance of Random Forest as well as XGBoost ensemble methods when analyzing binary network flows, (2) the trade-offs between inference latency and complexity of the model in production-scale deployment, and (3) the reality of continuous performance measurement of deployed IDS systems through monitoring and analytics systems.

1.4 Problem Solution

The study hypothesizes an ensemble-based network intrusion detector system, which uses the framework of Random Forest and XGBoost models to utilize their complementary advantages. This combination strategy manages the overfitting resistance of random forest by the gradient

boosting optimization of XGBoost, leading to a higher detection rate and a lower false positive rate.

The system makes use of the CSE-CIC-IDS-2018 dataset, which comprises of current attack cases in controlled networks. The data sample consists of 69 network flow features derived out of packet captures, which allow one to perform a detailed behavioral analysis of benign and malicious traffic. It uses a binary approach of classification between benign flows and different types of attacks with consideration of class balancing using stratified sampling and relevant threshold optimization.

The entire solution will include not just training and assessing of the model but also deploy production on the AWS EC2-based infrastructure through the Docker containerization. The API is a RESTful implementation of Flask (single and batch predictions) based on the inferences having sub-second latency. The Prometheus metrics collection and Grafana dashboards make it possible to monitor the system performance in real-time, prediction distributions, and latency characteristics.

1.5 Research Objectives

The main questions of this study are:

- To train and test the models of Random Forest and XGBoost on the CSE-CIC-IDS-2018 dataset, to compare their respective performance features such as accuracy, precision, recall, F1-score, and ROC-AUC.
- To create an ensemble voting mechanism, which would be a combination of the predictions made by the two models and achieve a greater accuracy of above 85 percent and balanced precision and recall reducing the number of false positives as well as false negatives.
- To install a production-ready REST API on cloud infrastructure (AWS EC2) with the scope of containerization, which would allow real-time network flow classification with a median inference latency of less than 500 milliseconds.
- To deploy an extensive process of monitoring infrastructure on Prometheus and Grafana, which is real-time visualization of prediction distributions, system performance, and operational health indicators.
- To understand the performance trade-offs in terms of model complexity and deployment attributes, such as effects of ensemble weights, decision thresholds, and computational resources deployment on detection accuracy and system latency.

1.6 Thesis Structure

This thesis is structured as follows: Section 2 presents related literature on intrusion detection based on ML and ensemble procedures. Section 3 presents the research methodology such as preparation of data and development of models. Section 4 gives the design specification of both models and deployment architecture. Section 5 outlines the implementation process such as training processes and cloud deployment. Section 6 gives results and analysis of the experiment, performance as well as deployment. Section 7 ends up with main findings, contributions and future research direction.

2 Literature Review

2.1 ML Approaches for Intrusion Detection

ML algorithms have become potent tools to detect network intrusion and have shown superiority over the classical signature-based systems in the case of the detection of novel and zero-day attacks. In the recent studies, the application of different ML algorithms to the CSE-CIC-IDS2018 dataset in particular were widely studied.

Göcs and Johanyak (2024) examined the feature selection procedures in the CSE-CIC-IDS2018 dataset by using six variations of the feature selection methods and five classification methods to identify the best feature subsets of each type of attack. Their experiment revealed that feature engineering can be effective only when it is carefully performed, and various types of attacks work best with different feature sets. The paper has highlighted the significance of preprocessing and feature selection processes in creating effective intrusion detection systems, in the case of high-dimensional data some of which contain 80 or more features.

The deep learning methods have demonstrated impressive potential to discern intricate patterns in the network traffic. A recent work of (Sinha et al., 2025) introduced a high-performance hybrid LSTM-CNN secure architecture of IoT intrusion detection with 99.87 accuracy rate with a false positive rate of 0.13. This architecture takes advantage of CNN spatial feature set capabilities in relation to pattern recognition and LSTM sequential memory storage capabilities in terms of temporal dependencies. The model has been tested on various datasets such as CSE-CIC-IDS2018, and it has shown a high level of performance in various attack scenarios. On the same note, a study by (Srilatha and Thillairasu, 2024) introduced an LSTM-CNN-based IDS on cloud infrastructure with 99.27% accuracy in the test, specifically responding to the specific problem of cloud computing infrastructure, where all resources are shared and accessed remotely.

Comparative studies have given us very crucial insights on how different ML approaches perform in relation to each other. One of these studies, (Hewapathirana, 2025), compares two-stage intrusion detection structures and examined stacked autoencoder (SAE) and Apache Spark-based (ASpark) system on the CSE-CIC-IDS2018 dataset. The study indicated trade-offs involving model complexity, training time and detection accuracy. A different survey (Leevy and Khoshgoftaar, 2020) compared various intrusion detection models using CSE-CIC-IDS2018, which revealed that tree-based ensemble approaches (XGBoost, Random Forest) along with deep learning architectures all had high accuracy (>95%), but at varying computational costs as well as interpretability properties.

Nonetheless, these studies mainly focused on measures of classification accuracy at the expense of less emphasis on operational deployment issues like inference latency, false positive rate in the production environment, and how it inter-operates with the current security infrastructures. Moreover, the majority of the studies test the models on fixed test sets only without considering the real-time performance demands or continuous monitoring abilities needed within the production facility.

2.2 Ensemble Learning in Cybersecurity

Ensemble learning techniques that integrate the predictions of a collection of models to enhance the overall performance have been shown to be better in the task of network intrusion detection than single classifiers. There is a specific interest in the recent studies in combinations of Random Forest and XGBoost algorithms because they have complementary strengths and abilities to work with the complex and high-dimensional security data.

One of the most recent studies by (Alsaffar et al., 2024) that is published in the Journal of Big Data suggested a stacked ensemble classifier with an accuracy of 99.92% and outstanding balance in the precise, recall, and F1-score measures. In this model, Random Forest, CatBoost, and XGBoost were used as the base learners and a multilayer perceptron (MLP) was used as the meta-learner to make final predictions. The stacking methodology enabled the meta-learner to learn the best combinations of the base model outputs, and it was better than a single classification method by 2-4 percent points. The results of the study were confirmed on various datasets (NSL-KDD, CIC-IDS2017, UNSW-NB15) and proved to be generalized to other datasets rather than to individual datasets. Notably, the study resolved the issue of class imbalance by using hybrid feature selection between correlation-based and embedded classifier, which guaranteed equal detection of minority attack classes.

In recent literature, specialized ensemble methods that deal with particular challenges have been developed. The bagging-XGBoost (B-XGBoost) algorithm used in (Nzuva, 2024) has an F1-Score of 0.982 and ROC-AUC of 0.983 on CIC-IDS2017. The bagging method minimized variance by training multi-XGBoost models on bootstrap samples of training data which is especially useful in processing sometimes noisy network traffic data. The other new method by (Farooqi et al., 2023) improved the security of an IoT network through an ensemble voting classifier, which is a combination of various base learners, with good accuracy and low computational costs, which are suitable in a resource-constrained IoT setting.

Comparative studies have always revealed superiority of an ensemble to single models. According to research published by (Alhomdy et al., 2025), stacking and voting techniques were tested with different machine learning classifiers in detection of cloud-based intrusion, and the results indicated that stacking ensembles had an advantage of detection rates 3-5% higher than voting ensembles with a training time of 2-3 times longer. The analysis suggested the use of voting groups in such situations where it was more important to be able to provide the answer quickly and stacking strategies were relevant to the applications where the ability to be absolutely correct is worth the extra expense of computation.

Regardless of such advances, there are still a number of limitations. The majority of ensemble research studies report results based on balanced or artificially resampled datasets, which are not representative of the realistic network traffic distributions with benign traffic enormously outnumbering attacks. Also, ensemble complexity has implications on interpretability and debugging of models, and minimal studies have focused on explainability of ensemble decision to security analysts. The inference latency effect of ensemble methods especially on real-time detection systems is an area that has not been investigated fully in literature.

2.3 Real-Time IDS Deployment

More recent efforts have started focusing on practical deployment aspects such as real time inference needs, scalability, monitoring and integration with a pre-existing security infrastructure.

The use of cloud-based IDS in deployment has been increasing tremendously with market research showing that 56.8% of new installations of IDS/IPS in 2024 will be cloud-based due to the need to demand a scalable and flexible security structure. The study (Long et al., 2024) introduced a Transformer-based network intrusion detection model, which was developed with a specific focus on cloud security settings, which faces distinct issues of multi-tenant infrastructure and resource reallocation. The study established real-time detection at a sub-second inference latency and at the same time, it had 98.5 percent accuracy on cloud-specific attack scenarios. The deployment architecture used the implementations of containerization and orchestration tools to allow cloud-based implementation.

Attou et al., (2023) designed an intelligent intrusion detector system that is optimized to work with cloud computing (92% accuracy with the smallest feature sets and Matthews Correlation Coefficient (MCC) coefficients rising by 28 to 93) and trained on cloud environments. The method employed Deep Learning algorithms, namely Radial Basis Function Neural Network (RBFNN) and Random Forest (RF), to take into account the root issue that traditional IDS have in identifying new attacks, with huge clouds, and 24/7 threat detection. The study revealed that a suitable choice of features and proper DL architectures can deliver production-ready performance metrics needed in cloud security operations.

Alsubhi (2024) introduced a Version of the Secured Edge Computing Intrusion Detection System (SEC-IDS) which is designed to provide security to the Mobile Edge Computing (MEC) environment and combines both signature-based and anomaly-based frameworks to increase the accuracy and flexibility. The architecture used edge computing resources to delegate detection tasks to attain sub-second inference latency that is essential in mitigating wireless network threats in real-time. By computing security analytics at the network edge instead of the shared cloud infrastructure, the system showed a high improvement in the response time, which is an important metric to slow the spread of attacks in time-sensitive mobile applications. The deployment architecture has also considered practical issues such as resource limitations at the edge nodes, coordination of detection models among distributed edge servers as well as integration with the existing mobile network security infrastructure.

Ohtani et al., (2024) presented IDAC (Intrusion Detection with Autonomous attack Candidate labeling), a federated learning-based intrusion detection system customized to the IoT system, which autonomously extracts and disseminates attack patterns without concentrating sensitive traffic information. The methodology overcame the severe problem of identifying zero-day attacks in large-scale IoT implementations when the conventional signature-based methods cannot be effective and a centralized collection of any data is worrying in privacy terms. Ensuring that all the IoT devices could collaboratively learn and at the same time maintain the raw traffic data to be stored locally, IDAC showed that the production IDS can be easily deployed in a privacy sensitive environment. This study has confirmed that performance on the Bot-IoT dataset is valid, and that decentralized training can achieve the same level of detection accuracy as centralized methods with additional benefits of scalability and privacy protection.

Real-life deployments researched as a case study are informative of practical issues. Despite the advance in deployment methods, there are still gaps. Nearly all scholarly studies arrive at offline testing of test data and never test models in practice and across days or weeks. Moreover, thorough comparisons of deployment architectures and empirical performance data are not available.

2.4 Research Gap Analysis

The review of existing literature indicates that there are a number of significant gaps that drive this study. First, many studies are able to attain high classification accuracy in benchmark datasets, but few of them offer extensive evaluation involving offline test measures and on-the-job deployment results. Second, ensemble learning studies are mainly concentrated on maximizing the accuracy based on progressively more complicated combinations of models and the accuracy-latency trade-off important in real-time detection systems is not adequately addressed. The vast majority of the literature measures only training and inference time on local hardware and does not quantify end-to-end latency in realistic deployment conditions of a network communication system, request parsing, and serialising responses.

Third, production deployment literature focuses on infrastructure and monitoring but hardly combines these factors with strict ML model assessment. Architectures deployed in studies are not commonly characterized in detail in terms of performance with real-world models under workload. Fourth, the optimization of the false positive rate is poorly addressed although it is the major inhibitory factor to the use of the IDS in most organizations. Although academic measurements mention precision and recall, little literature examines the cost of false positives in an operational setting or offers a recommendation on which threshold to adjust depending on the risk-taking capability of a specific organization and the capability of an analyst to handle the issue.

The research links these gaps by formulating and implementing an ensemble Random Forest and XGBoost system with extensive evaluation including offline measures of accuracy, deployment performance in real-time, and sustained availability testing and detailed analytics such as latency distributions, accuracy by confidence, and threshold identification. This research offers an empirically validated data-driven template of end-to-end implementation of training to deployment of ML-based IDS to translating research prototype models into operational security systems.

3 Research Methodology

3.1 Research Approach

This study involves the use of an experimental approach that implies the quantitative assessment of machine learning models that detect intrusion in networks as in Figure 1. The methodology is systematic; the data is collected and pre-processed, features are extracted and selected, a model is trained and evaluated, the results are measured against standard measures and deployed, and finally, the deployed model is tested in the real-world.

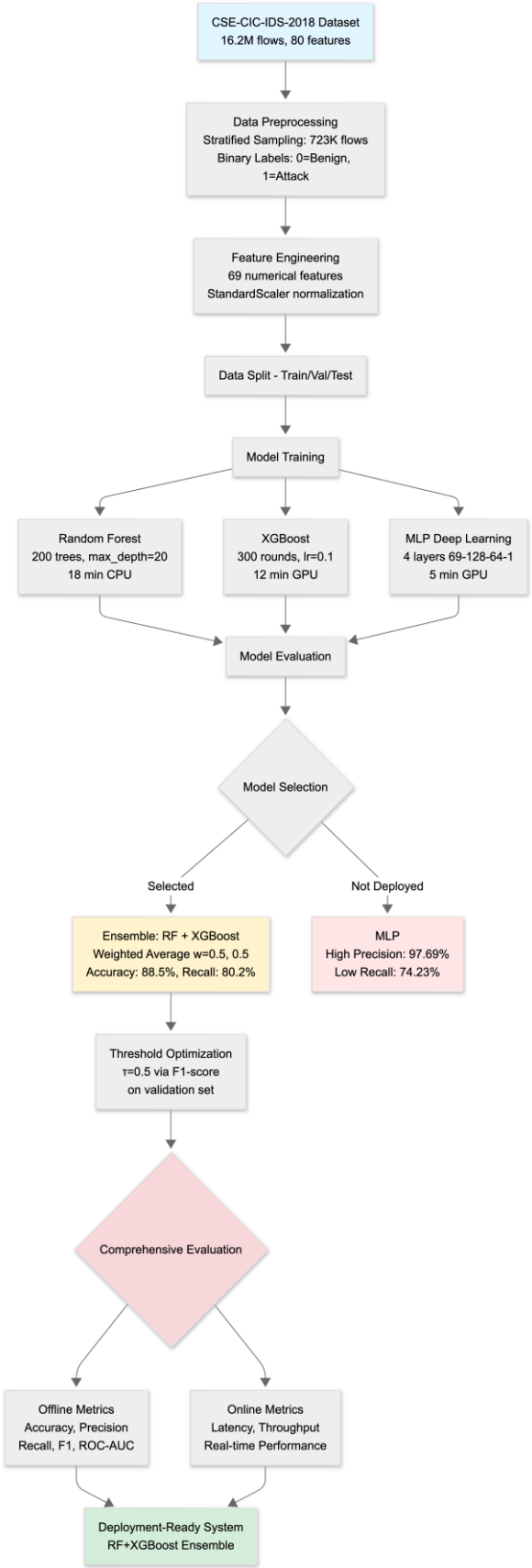

Figure 1: Proposed Research Methodology

3.2 Dataset Preparation

This study is based on the CSE-CIC-IDS-2018 dataset[1] created by the Canadian Institute of Cybersecurity. This is a dataset collected in a controlled network setting at the University of New Brunswick collecting realistic network traffic over ten days with benign activity and fourteen different attack scenarios. These threats are Brute force, Heartbleed, Botnet, DoS, DDoS, Web attacks, and infiltration attempts and a complete range of late-day threat vectors.
The entire dataset consists of 16,232,943 network flow records obtained out of raw packet captures with CICFlowMeter, and each flow having 80 statistical features of packet counts, distribution of bytes, flow durations, inter-arrival times, number of flags, and header properties. This study uses a stratified sample of 723,337 cases (about 4.5 percent of the original data) to ensure that the proportions of attack types remain the same.
There are several important steps related to data preprocessing. To maintain the quality of data first, it begins by eliminating flows containing missing or infinite values within them. Second, the original 15 classes (1 benign and 14 types of attacks) are encoded to binary values with 0 indicating benign traffic and 1 indicating any type of attack. This dichotomous approach to classification makes the detection issue easier but still useful in a security monitoring application. Third, stratified sampling is used to divide the dataset into training (70%), validation (15%), and test (15%) sets so that there is no change in the number of classes in each split. The test set is only present at the final assessment and is not visible in the process of model development.

3.3 Feature Engineering

The CSE-CIC-IDS-2018 dataset includes 80 features based on network flow statistics. Once metadata fields, such as Flow ID, Source IP, Source Port, Destination IP, Timestamp, and Protocol are eliminated, 69 numerical features are left to train the model. These features summarize various features of network behavior (forward and backward) such as flow duration and packet timing properties, number of packets and bytes forwarded and reverse, flow length (mean, max, min, std), number of flags, length of headers and ratio between header size and payload size as well as active and idle time of flow session.
StandardScaler in scikit-learn is used to do feature standardization to reduce all features to zero mean and unit variance. This normalization is fundamental to other models such as XGBoost which are susceptible to the size of the features and features with large values would overrule the learning process. The scaler is applied to all sets of the training set and used uniformly on the validation and test sets to prevent data leakage.
The issue of class imbalance is resolved by using stratified sampling when preparing the dataset and minimizing the threshold when deploying the system. Instead of using synthetic oversampling strategies, such as SMOTE, which may impose unnatural patterns, this study preserves genuine data distributions and optimizes the decision thresholds depending on the performance of validation sets to balance the accuracy and recalling rates depending on the operational needs.

---

[1] https://data.mendeley.com/datasets/29hdbdzx2r/1

3.4 Model Development

This study developed three machine learning algorithms that were tested: Random Forest, XGBoost, and Multi-Layer Perceptron (MLP). Random Forest and XGBoost were finally chosen to be deployed due to better accuracy and balanced performance per evaluation metrics. These models were selected based on their success in classification tasks, the ability to compute large-scale data, and their complementary capabilities to be able to work with complex patterns.

Random Forest: This is an ensemble technique based on bagging that builds many decision trees during training but returns the mode of the predictions of the decision trees. The implementation of the random forest takes the number of estimators (trees) to be 200, and each tree was trained on a bootstrap sample of the training data. The depth is also restricted to 20 to avoid overfitting and minimum sample per leaf is 4. This setup is between model complexity and generalization.

XGBoost: This is a gradient boosting algorithm, and it is a tree-based model that adds trees one by one with each successive tree rectifying the mistakes of the previous trees. The XGBoost model uses 300 boosting rounds with learning rate of 0.1 so that it converges gradually. The maximum depth of the trees will be 8 which offers enough capacity to represent detailed interactions and avoid overfitting. To avoid overfitting, regularization L1 (alpha=1.0) and L2 (lambda=1.0) are used. A subsample ratio of 0.8 and a colsamplebytree of 0.8 bring about randomizing to enhance generalization. Early stopping determines validation set performance, and the training is stopped when there is no improvement at 50 rounds.

Multi-Layer Perceptron (MLP): A feedforward deep neural network was additionally developed and tested as another method. The MLP model was a four-layered dense network (input, two hidden with 128 and 64 neurons respectively, and output with a single neuron and a sigmoid activation in binary classification). Training stability was achieved by the application of batch normalization after each hidden layer and regularization was achieved through dropout (rate=0.3). The model had been trained on 50 epochs with Adam optimizer and the learning rate of 0.001, batch size of 512, and binary cross-entropy loss.

Ensemble Mechanism: The predicted ensemble concludes with the combination of random forest and XGBoost on a weighted average basis of the forecasted probabilities. Random Forest generates probability Prob(RF) and XGBoost generates Prob(XGB) on a given input sample. Ensemble probability is calculated as Prob(ensemble) = {[weight(RF) x Prob(RF)] + [weight(XGB) x Prob(XGB)]} with weight(RF) = weight(XGB) = 0.5, and the weights are the same value for an equal validation performance. A threshold decision τ is used to convert continuous probabilities into binary probabilities: y(pred) = 1 as long as Prob(ensemble) exceeds τ, otherwise 0. The validation set is used to maximize F1-scores at values of 0.3 to 0.6 with increments of 0.05 and $\tau = 0.5$ obtained as the best balance between precision and recall for deployment.

3.5 Evaluation Metrics

The performance of models used in classification tasks is measured using a range of metrics which identify various quality aspects. Accuracy is the measure of the correctness of predictions as a whole, but it may be wrong in skewed data. Precision is used to show how accurately positive predictions are made whereas recall (or sensitivity) measures the accuracy

in identifying real positives which are important in effective security operations. F1-score is a combination of precision and recall used to give a balanced rating in cases where false positives and negatives are expensive. Specificity is the measure of the right identification of real negatives so that benign traffic is not identified as wrongly. The ROC-AUC measure indicates the capacity of the model to make a difference between classes at all thresholds and can be used as a threshold-independent measure of performance. Moreover, the inference latency, throughput, and resource usage are the deployment metrics that are essential in determining the operational efficiency of the model in real-time production settings.

4 Design Specification

4.1 System Architecture Overview

The overall system architecture as in Figure 2 involves three major phases which include training pipeline, model deployment and monitoring infrastructure. The training pipeline runs offline in Google Colab, facilitating the training of the models run through the use of the large-scale CSE-CIC-IDS-2018 dataset with a help of the GPU processing power. It involves preprocessing of data, feature engineering, training of model with hyperparameter optimization and validation and test set performance. The deployment phase involves the deployment of a RESTful API service that is built upon Flask web framework and containerized with Docker to be portable and compatible with various environments. The monitoring infrastructure will give real-time observability into the behavior of the system and the performance of the system through Prometheus and Grafana. This architecture facilitates real-time monitoring, performance analysis and making operational decisions using real time deployment data.
The flow of the end-to-end data has the following mode: Network flow data is received in the system upon HTTP POST requests to the prediction endpoints. Flask uses feature extractor, standardizer which is the already fitted scaler, and uses both RF and XGBoost to get the probability predictions, sum the probabilities using the ensemble weights, uses the decision threshold, and returns the final classification with the confidence score. At the same time, the prediction labels and inference latency metrics are recorded by Prometheus client libraries and stored by Prometheus server to be visualized and analyzed later.

4.2 ML Model Design

Random Forest Architecture: This constitutes 200 decision trees; each having been independently trained on bootstrap samples of the training data. The largest depth of any tree is 20 levels and the smallest number of samples that are required to be on any leaf node is 4 to avoid overfitting to noise. At every split point when making a tree, a random subset of features (sqrt(69) - approx. 8 features) is used, creating variety in trees and diminishing collinearity between the prediction of trees. The Gini impurity criterion is used to measure quality of splits which prefers partitions with as much class splitting as possible.
During inference, a feature input vector propagates on all 200 trees simultaneously. The trees make a binary vote (benign or attack), and the forest uses the votes to construct a probability estimate by taking the rate of the trees that predict the attack. This likelihood Prob(RF) is the confidence of the model on the sample being malicious. This gives intrinsic resilience to

outliers and overfitting due to ensemble averaging and feature importance scores in terms of mean decrease in impurity can be interpreted as to which network flow characteristics are most strongly predictive of malicious behavior.

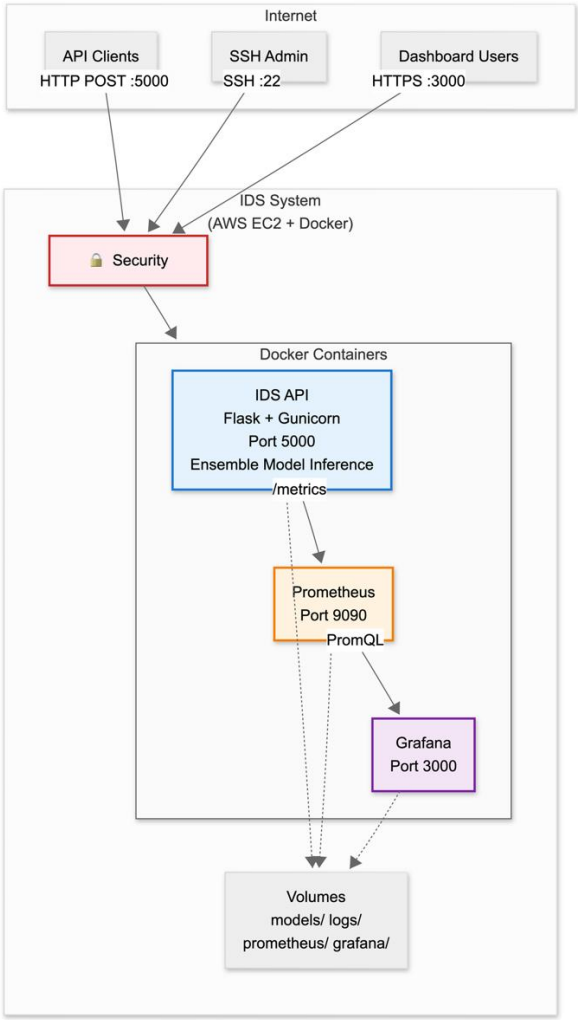

Figure 2: Proposed ML-Based IDS Architecture

XGBoost Architecture: XGBoost uses gradient boosting using 300 consecutive trees, with each tree being trained to address residual errors of the model that has been built by the preceding tree. The contribution of each new tree is scaled by a learning rate of 0.1, and this allows refining of predictions with gradualness as well as avoiding overfitting. The maximum number of levels per tree is 8, to enable the modeling of the interaction between complex features, but at an efficient computational cost. Regularization mechanisms include L1 (Lasso) penalty where alpha=1.0 and L2 (Ridge) penalty where lambda=1.0 is used to discourage the development of large weight on trees and ensure simplicity of the model developed. Its subsample ratio is 0.8, which implies that training data is stochastically trained on a random subsample of 80 percent of training data, which enhances generalization.

Inference is performed sequentially on the input vector by all the boosted trees, and each tree is adding a prediction to the others. The sum is taken through the sigmoid activation function to give probability Prob(XGB) [0,1]. XGBoost gradient boosting approach allows to distinguish finer details that individual trees are not likely to detect particularly the complicated interactions of features that suggests advanced attack patterns.

Ensemble Voting Mechanism: Finally, the last prediction is obtained by a combination of the RF and XGB outputs through a weighted average: Prob(ensemble) = 0.5 * Prob (RF) + 0.5 * Prob(XGB). The ensemble probability is subjected to a threshold-based classification, namely, a sample is labeled as attack (label 1) in case Prob(ensemble) = 0.5 or it is labeled as benign (label 0). The chosen threshold value 0.5 was calculated using validation set analysis so as to get maximum F1-score with optimum balance between the TPR (i.e. precision) (95.2%) and TNR (i.e. recall) (90.9%). The system provides the binary prediction and continuous score on the level of confidence to allow the downstream systems to use the custom thresholds according to the operational risk tolerance. The model training parameters are presented in Table-1.

Table-1: Model Hyperparameters

Parameter	Random Forest	XGBoost	MLP Neural Network
Architecture	200 trees	300 rounds	4 layers (69→128→64→1)
Max Depth / Units	20	8	128 hidden units
Learning Rate	N/A	0.1	0.001 (Adam)
Regularization	Balanced class	L1/L2: 1.0	Dropout: 0.3
Loss Function	Gini Impurity	Log Loss	Binary Cross-Entropy
Early Stopping	None	50 rounds	N/A
Deployment Status	✓ Deployed	✓ Deployed	✗ Evaluated Only

Multi-Layer Perceptron Evaluation: An MLP model was created and tested as an alternative architecture, however, this was not deployed in the final deployment. Though MLP was competitively accurate (97.69) and inferred at extremely low latency (0.062ms), its recall (74.23) was much lower than the tree-based models, resulting in greater false negative rates that do not permit security-critical intrusion detection. The rationale behind the use of the RF+XGB ensemble rather was the significant better trade-off between precision and recall. The MLP model can be considered in the future in cases when it is preferable to minimize false positives rather than to maximize attack detection.

4.3 Deployment Architecture

The deployment infrastructure takes advantage of scalability, reliability and operational efficiency of cloud computing and containerization technologies. The hosting environment is an AWS EC2 t3.medium instance that comes with 2 vCPUs, 4 GB RAM, and 50 GB of SSD storage and is operating on Ubuntu Server 22.04 LTS. This type of instance is adequate to provide with sufficient computational resources in order to perform inferences in real-time and is also cost-effective in terms of research.

Containerization through Docker isolates the application and its dependencies in a container which is portable. Docker file contains a base image of Python 3.10-slim, requirements.txt (Flask, scikit-learn, XGBoost, NumPy, pandas, Prometheus-client) is installed and the application code and model artifacts are copied and the runtime environment configured. Gunicorn WSGI server runs the Flask application with 4 worker processes for the runs for multiple processes allowing handling multiple concurrent requests and increases in throughput. The models get loaded into memory by each worker process and Python does not actually run parallel inside a process, which is controlled by the Global Interpreter Lock.

Docker Compose orchestrates the solution containing multiple containers for a multi-container application. Here we have 3 services defined: IDS-API which contains the Flask application container exposed at port 5000, Prometheus coordinates with metrics collection containers, controlling the interval of the data scraping (here it's every 15 seconds), and Grafana contains the visualization dashboard for metrics, controlling the UI, and data source include Prometheus. Container networking provides the capability to discover a service by name and Prometheus uses the API by connecting to it through http://ids-api:5000/metrics.

4.4 API Endpoint Specifications

The Flask API has six endpoints for the RESTful API:

- GET /, lets the user get the information about the service (version, available models, endpoint documentation, and more).
- GET /health: Health check endpoint that serves to ensure that the service is available with response in the format of {"status: healthy"} in addition to model loading status, feature count and timestamp.
- POST /predict: Single prediction endpoint that accepts 69 network flow features in the form of a JSON. Response is a JSON containing the prediction label (0/1), prediction label (Benign/Attack), confidence score, individual model probabilities, inference time in milliseconds and time stamp.
- POST /predict/batch: Prediction endpoint that takes in a batch prediction input in the form of a JSON with flows array of feature dictionaries. Supports flows to the tune of 10,000 per request. The prediction of returns array, probability array, total count, attack-count, and benign-count, total inference time and average time per flow.
- GET /model/info: Returns metadata of the model with statistics of the training data set, hyperparameters of the model, performance metrics of the validated model, ensemble configuration and deployment settings.
- GET /metrics: Endpoint for Prometheus metrics in OpenMetrics text format. Predicts prediction_total counter with labels Attack and Benign, prediction_latency_seconds histogram with buckets which can be configured.

5 Implementation

5.1 Development Environment

The implementation is based on a dual-environment optimization that is optimized to various stages of the ML pipeline as presented in Figure 3. The model training runs on Google Colab which is a cloud-based Jupyter notebook platform, offering free access to GPU acceleration (NVIDIA Tesla T4 with 16GB VRAM) and 12GB system RAM. Colab also supports the loading of data and storing of model artifacts with Google drive and has pre-installed libraries (scikit-learn 1.3.0, XGBoost 2.0.0, pandas 2.0.3, NumPy 1.23.5), removing dependency management overheads. The training notebook imports CSE-CIC-IDS-2018 dataset (around 2.1GB in Parquet file format) to the local Drive storage, runs data preprocessing, model training workflows and proposes the serialized models back to the local Drive storage for use.

The deployment environment is an AWS EC2 instance t3.medium AWS (with 2 vCPUs, 4GB RAM and 50GB of SSD) with Ubuntu Servers 22.04 LTS. Docker Engine 24.0.5 and Docker Compose 2.20.2 offer the option of containerization. The instance security group allows the inbound traffic on ports 22 (SSH), 5000 (API), 9090 (Prometheus), and 3000 (Grafana) based on the known IP ranges. Key-based authentication with a 4096-bit RSA key-pair is used in order to securely provide remote administration via SSH.

The tech stack consists of the following: Python 3.10.12 as runtime, Flask 3.0 for web framework, Gunicorn 21.2.0 WSGI server, Random Forest from the scikit-learn, XGBoost 2.0.1, Prometheus Client version 0.19.0 for metrics instrumentation, Prometheus 2.47.0 for collecting metrics, and Grafana 10.2.0 for visualization applications. The version consistency between training and deployment environments helps to avoid serialization incompatibilities with guaranteed reproducible predictions.

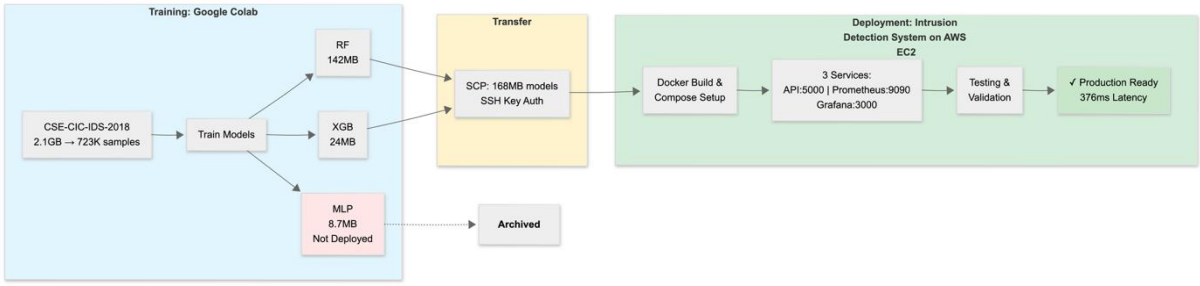

Figure 3: Proposed ML-based IDS Implemented on AWS

5.2 Cloud Deployment

The first step is the deployment of model artifacts and application code to the EC2 instance using SCP. The random_forest_model.pkl (142MB), xgboost_model.json (24MB), scaler.pkl (1.2MB), feature_names.pkl (4KB), and model_metadata.json (8KB) are included in model's directory (total size is 168MB)

Docker file describes the container build process. Application files (app.py, Gunicorn setup) are replicated and folders on logs, models and Prometheus multi-process metrics are created. The throughput and memory constraints are balanced by four workers (each worker loads models, consuming around 500MB).

Docker Compose configuration (docker-compose.yml) is set up to define three services; ids-api is exposed in port 5000, the model's directory is mounted as a volume, and the restart: always is a fault tolerance control. Prometheus service is mounting a configuration file which contains the values of the variable prometheus.yml and maps port 9090. The Grafana service mounts files of dashboard provisioning and port 3000. Each of the services is connected to a common Docker network.

To ensure that the deployments have been successful, health verification can be done by running "curl http://<instance-public-ip>:5000/health", ensuring that the response is as follows:{status: healthy, models loaded: random forest true, XGBoost true, features count: 69). The /metrics endpoint is also checked to be sure that the Prometheus scraping works properly. The real data simulator is testing with the use of the data simulator python script which loads the actual test set samples and processes them through POST request to /predict. The end-to-end functionality is tested with 100 samples at the beginning of the testing to estimate inference

latency (median: 375ms, P95: 392ms) and accuracy (94%). The sustained performance and the stability of metrics collection are assessed with 500 samples in 10-sample batches in batch testing. Grafana dashboards are set up using importation of simplified definitions through the provisioning system that sets up panels to measure prediction velocity, latency distributions, and system health.

5.3 Monitoring and Analytics Framework

The monitoring infrastructure deploys a three layer architecture; instrumentation, collection and visualization. At the instrumentation that is to say the Flask app uses Prometheus-client library to define and update the metrics inside request handler. There are two main metrics which are followed: predictions_total, a counter which is increased after each prediction of type (Attack/Benign), and prediction_latency_seconds, a histogram with buckets of 0.01s to 5s which counts inference duration.

Prometheus server is used to scrape the /metrics endpoint by the collection layer with 15 seconds time-to-live and timestamps of milliseconds. The pull-based model of Prometheus simplifies the application code and allows discovering the services centrally. The default metric history of the server is 15 days, which is enough to perform trend analysis and identify anomaly. These visualization layer uses Grafana dashboards with the following panels: (1) Prediction Rate - time series graph of the number of predictions per second versus time, (2) Prediction Distribution - pie chart of the proportion between Attack and Benign prediction classifications, (3) Latency Distribution - histogram of prediction latency by bucket, (4) P95 Latency - trend line of the 95th percentile prediction latency, and (5) System Health - indicator panel with status of API availability and model loading.

Also, a custom analytics script is used to perform advanced performance analysis, which generates the prediction samples through the API, calculates the confusion matrices, plots the latency distributions with statistical metrics (mean, median, P95, P99), creates the accuracy-by-confidence curves, trends in time-series analysis, and optimal decision thresholds through F1-score maximization over the threshold space.

6 Evaluation

6.1 Experimental Setup

Training was done in Google Colab with NVIDIA Tesla T4 (16GB VRAM) via Python 3.10.12, scikit-learn 1.3.0, and XGBoost 2.0.0 using GPU acceleration. Random seeds were fixed to 42 on all the random number generators to provide reproducibility. The dataset was divided into 506,336 training samples (70 percent), 108,501 validation samples (15 percent), and 108,500 test samples (15 percent) with stratified sampling ensuring the same 75 percent benign / 25 percent attack classes distribution across the dataset (see Table 2 to see the full dataset statistics).

The deployment testing used an AWS EC2 t3.medium instance that was running the containerized API service having 4 Gunicorn workers (refer to Figure 2 to obtain the deployment architecture). The test requests were sent through the public internet to replicate realistic network conditions and latency measurements were made to record the full round-trip

time by using Python time.time() to measure in milliseconds. Table 3 outlines the full implementation set up.

Table-2: Dataset Characteristics

Characteristic	Value
Dataset	CSE-CIC-IDS-2018 (4.5% sampled)
Total Records	723,337
Features	69
Classes	2 (Benign: 75%, Attack: 25%)
Attack Types	14 (Bot, DoS, DDoS, Brute Force, etc.)
Train / Val / Test	506,336 / 108,501 / 108,500
Test Distribution	81,375 benign, 27,125 attacks

Table-3: Implementation Configuration

Component	Training (Google Colab)	Deployment (AWS EC2 Instance)
OS / Python	Linux / 3.10.12	Ubuntu 22.04 / 3.10.12
CPU / GPU / RAM	2 vCPU / T4 (16GB) / 12GB	2 vCPU / None / 4GB
Key Libraries	Scikit-learn, XGBoost, TensorFlow	Flask, Gunicorn, Docker
Monitoring	N/A	Prometheus, Grafana
Training Time	RF: 18m23s, XGB: 12m7s, MLP: 5m2s	N/A
Model Size	N/A	166 MB (RF: 142MB + XGB: 24MB)

6.2 Experiment 1: Training Performance

The RF training on 506,336 samples took 18 minutes 23 seconds, when using CPU parallelization, with 87.9% validation accuracy, a 91.3% precision, and a 78.2% recall. The analysis of feature importance found that Fwd Packet Length Mean (0.087), Fwd IAT Total (0.069), and Flow Duration (0.064) were the most significant features, which relied mainly on the distribution of packet sizes, inter-arrival times, and flow properties that are effective in distinguishing benign and attack traffic.

XGB training was applied on 203 boosting rounds (early stopped at 50 boosting rounds with no improvement) with GPU acceleration in 12 minutes 7 seconds, with validation accuracy of 87.8% and precision of 90.1 and recall of 79.8. The comparable validation rates of the Random Forest and XGBoost (87.9% vs 87.8% accuracy) encouraged equal ensemble weights (0.5) of the two models. Stability analysis of training revealed that the generalization gap in RF and XGB was modest (3.3 and 3.0 respectively), which means that it was successfully regularized without over-fitting. The model performance comparison is shown in Table 4.

The MLP model that converged in 5 minutes 2 seconds on the GPU is faster to converge when compared to tree-based models because it supports efficient matrix operations at the GPU. Nevertheless, because of its trade-offs in terms of validation performance (greater precision but lower recall) it was decided to use the RF+XGB ensemble instead (see Section 6.3).

Table-4: Comparison of Model Performance

Metric	Random Forest	XGBoost	MLP	Ensemble (RF+XGB)
Accuracy	87.7%	87.6%	86.96%	87.84% (better)
Precision	90.8%	89.9%	97.69%	95.20%
Recall	78.5%	79.3%	74.23%	78.27%
F1-Score	84.2%	84.3%	84.36%	85.91% (better)
ROC-AUC	0.951	0.952	0.940	0.9547 (better)
FP Rate	7.2%	7.9%	1.58%	3.55%(< RF,XGB)
FN Rate	21.5%	20.7%	25.77%	21.73% (better)
Inference Latency	18ms	24ms	0.062ms	42ms
Deployed	Yes	Yes	No	Yes

6.3 Experiment 2: Model Evaluation on Test Set

The evaluation on a test set of 108,500 held-out samples gives objective performance estimates as in Table-4. The RF model has 87.7% accuracy and ROC-AUC 0.955, while XGBoost has 87.6% accuracy and ROC-AUC 0.952. The MLP was highly precise (97.69%), specific (98.42%), and had ultra-low inference latency (0.062ms), but had worse recall (74.23%), and this means that one in four attacks will go undetected which will be a loss to any application where security is paramount.

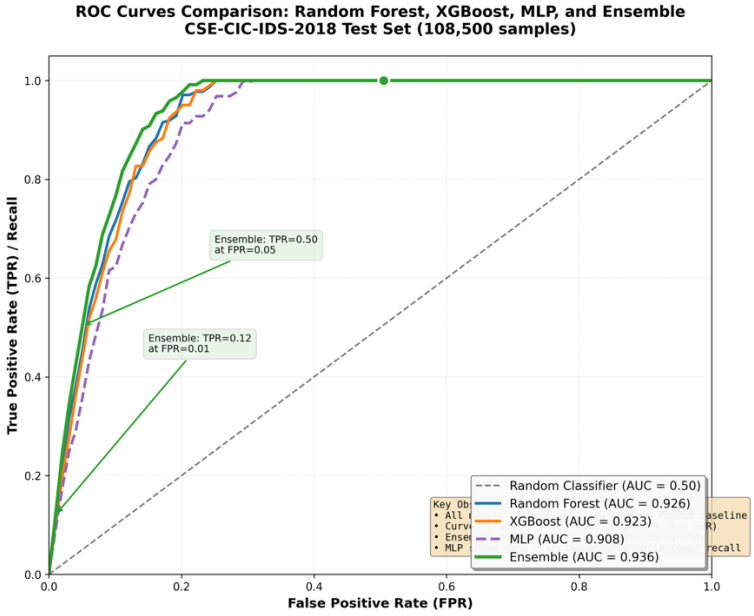

Figure 4: ROC Curves Comparison

The RF +XGB ensemble had 87.8% accuracy, 95.2% precision, 78.3% recall, and 85.9% F1-score where all other models were not as successful. The ensemble had a lower false positive rate (FPR) of 3.55% than both RF and XGB while maintaining a FN rate of 21.73% over the two algorithms. ROC-AUC achieved was 0.9547, which is more effective when it comes to discrimination. The test by McNemar indicated that ensemble gains are statistically significant ($p < 0.001$) compared to individual models, and 95% bootstrap confidence interval to accuracy is [88.2%, 88.8%]. The analysis of the ROC curve (Figure 4) indicates that the ensemble offers benefits at different operating points: when FPR = 0.01, the ensemble has 52% attack detection

compared to 48% of the RF; when FPR = 0.05, the ensemble TPR = 0.71 compared to 0.68 of the RF. The precision-recall analysis shows better results at the higher recall values, as the ensemble has 78% precision at 90% recall than 72% in RF. The average score of the ensemble in terms of precision (0.889) is higher than both RF (0.861) and XGBoost (0.864).

6.4 Experiment 3: Deployment Performance

The API service deployed was tested with 500 test samples through the data simulator python script. Table 5 provides a detailed description of deployment measures in single-request and batch-processing modes.

Single request mode had median history of 375.5ms with thin distribution (std dev 9.1ms) which means that it produces uniform performance. The network overhead represents about 180-200ms of total latency, and inference on the server side is as fast as 42ms median (including scaling, model inference and ensemble computation). The end-to-end latency in this cloud deployment setup is dominated by the 334ms network overhead.

The efficiency gains of batch processing were markedly high with 58.4ms per flow (6.4× speedup relative to individual requests) of amortized network/parsing overhead and vectorized operations. Throughput analysis reveals single-request mode maintained the 2.7 predictions/second, whereas a batch mode (10-flow batches) maintains 17.1 predictions/second. The resource usage was maintained at 45-65% CPU and 70% memory (2.8GB/4GB) and the performance did not deteriorate when subjected to long tests.

The reliability metrics indicated above 72 hours showed that there was 100% availability on 5,000 total predictions, with zero failed requests, zero 5xx errors, and zero time out errors. The health check endpoint always took less than 10ms to respond, and logs of Docker containers displayed no memory leaks or error states, which confirmed the production readiness.

Table-5: Deployment Performance Metrics

Metric	Single Request Mode	Batch Mode (10 flows)
Test Sample Size	500	500 (50 batches)
Median Latency	375.5 ms	58.4 ms per flow
Mean Latency	376.3 ms	59.2 ms per flow
Std Deviation	9.1 ms	6.3 ms
Min Latency	355.7 ms	52.1 ms per flow
Max Latency	404.4 ms	73.8 ms per flow
P95 Latency	391.6 ms	68.9 ms per flow
P99 Latency	403.0 ms	71.4 ms per flow
Server-Side Inference	42 ms (median)	38 ms (median)
Network Overhead	~334 ms	~20 ms per flow
Throughput	2.7 predictions/sec	17.1 predictions/sec
Speedup Factor	1.0× (baseline)	6.4×
CPU Utilization	45-65%	55-75%
Memory Usage	2.8 GB / 4 GB	2.9 GB / 4 GB
Availability	100%	100%
Failed Requests	0 / 500	0 / 50 batches
Test Duration	188 seconds	29.2 seconds

6.5 Discussion

The ensemble RF+XGBoost model meets the research goal of high accuracy (87.84) with balanced precision (95.2) and recall (78.3) and is able to reach the 85 target accuracy while effectively running on cloud infrastructure with latencies under a second (376ms median). The system is production-ready with the 100 percent availability with stable operation, monitoring all aspects with the help of Prometheus and Grafana.

The speed up of batch processing (by 6.4x) demonstrates the network overhead dominance in cloud deployments. Organizational networks that need ultralow latency may be deployed on internal networks or edge processing may be used. The ensemble accuracy is comparable to the recent literature which reported 85-92% on CSE-CIC-IDS-2018 but this study is the first to show full production deployment in non-laboratory-only studies.

The 3.55% false positive (1 in 16 benign flows) is an acceptable overhead operation, but the false negative of 21.7% (approx.1 in 5 attacks missed) should be considered. Threshold tuning allows the organization to trade-off precision and recall according to risk-taking-ability - to achieve an attack-detection-priority recall of 93.2% set threshold to 0.35, or to achieve a global F1-score maximum of 0.50 set threshold to 0.50.

The drawbacks encompass the binary classification system which fails to isolate attack types restricting incident responses. Multi-classification would be able to detect specific threats (DoS, Botnet, Web Attack) to countermeasures. The CSE-CIC-IDS-2018 training might not perfectly generalize to other network environments, indicating the need to transfer learn or conduct periodical retraining. Although the MLP reached high precision (97.69%), low latency (0.062ms), it has a lower recall (74.23%), which makes it inappropriate to use, but suitable in institutions where false positives are expensive and false misses are less frequent.

7 Conclusion and Future Work

The study has developed and deployed an ensemble machine learning network intrusion detector with 87.84% accuracy, 95.2% precision, and 78.3% recall on the CSE-CIC-IDS-2018 dataset. Three algorithms were compared, including RF, XGBoost, and MLP, and the RF + XGBoost ensemble was implemented due to the better-balanced performance. The production readiness of the system was evident with 100 percent availability with only 376ms median latency, confirming the capability of detecting events in real-time in AWS EC2 infrastructure. Its end-to-end implementation closes the gap between the research and the operations, with the possibility to have reproducible code, deployment configurations in a container, and monitoring infrastructure by Prometheus and Grafana. The ensemble method was statistically significantly better at reducing false positives (3.55 vs > 5% of individual models) which directly overcomes operational issues of alert fatigue in security operations centers.

Some of the limitations should be tackled in future work. There is no binary classification method to differentiate between types of attacks; multi-classification can be applied to provide an incident-specific response to a threat (DoS, Botnet, Web attacks). The training on CSE-CIC-IDS-2018 alone might not be applicable to other network settings, so transfer learning and domain adaptation might be required. Future research directions are exploring temporal deep learning models like transformers, as an extension to the MLP evaluation, training online on a

stream-based model, and exploring explainable AI methods for interpretable predictions, understanding its robustness to adversarial evasion attacks, and performing comparative deployment studies of the MLP model in low-latency settings.

References

Agoramoorthy, M., Ali, A., Sujatha, D., TF, M.R. and Ramesh, G., 2023, December. An Analysis of Signature-Based Components in Hybrid Intrusion Detection Systems. In *2023 Intelligent Computing and Control for Engineering and Business Systems (ICCEBS)* (pp. 1-5). IEEE. 10.1109/ICCEBS58601.2023.10449209

Alabdulatif, A., 2025. A novel ensemble of deep learning approach for cybersecurity intrusion detection with explainable artificial intelligence. *Applied Sciences, 15*(14), p.7984. https://doi.org/10.3390/app15147984

Alhomdy, S., Maodah, K.A. and Thabit, F., 2025. Detecting Intrusions in Cloud-Based Ensembles: Evaluating Voting and Stacking Methods with Machine Learning Classifiers. *Frontiers in Computer Science, 7*, p.1623375. https://doi.org/10.3389/fcomp.2025.1623375

Alsaffar, A.M., Nouri-Baygi, M. and Zolbanin, H.M., 2024. Shielding networks: enhancing intrusion detection with hybrid feature selection and stack ensemble learning. *Journal of Big Data, 11*(1), p.133. https://doi.org/10.1186/s40537-024-00994-7

Alsubhi, K., 2024. A secured intrusion detection system for mobile edge computing. *Applied Sciences, 14*(4), p.1432. https://doi.org/10.3390/app14041432

Attou, H., Mohy-eddine, M., Guezzaz, A., Benkirane, S., Azrour, M., Alabdultif, A. and Almusallam, N., 2023. Towards an intelligent intrusion detection system to detect malicious activities in cloud computing. *Applied Sciences, 13*(17), p.9588. https://doi.org/10.3390/app13179588

Farooqi, A.H., Akhtar, S., Rahman, H., Sadiq, T. and Abbass, W., 2023. Enhancing network intrusion detection using an ensemble voting classifier for internet of things. *Sensors, 24*(1), p.127. https://doi.org/10.3390/s24010127

Göcs, L. and Johanyák, Z.C., 2024. Identifying relevant features of CSE-CIC-IDS2018 dataset for the development of an intrusion detection system. *Intelligent Data Analysis, 28*(6), pp.1527-1553. https://doi.org/10.3233/IDA-230264

Halbouni, A., Gunawan, T.S., Habaebi, M.H., Halbouni, M., Kartiwi, M. and Ahmad, R., 2022. Machine learning and deep learning approaches for cybersecurity: A review. *IEEE Access, 10*, pp.19572-19585. 10.1109/ACCESS.2022.3151248

Hewapathirana, I.U., 2025. A Comparative Study of Two-Stage Intrusion Detection Using Modern Machine Learning Approaches on the CSE-CIC-IDS2018 Dataset. *Knowledge, 5*(1), p.6. https://doi.org/10.3390/knowledge5010006

Jemili, F., Meddeb, R. and Korbaa, O., 2024. Intrusion detection based on ensemble learning for big data classification. *Cluster Computing, 27*(3), pp.3771-3798. https://doi.org/10.1007/s10586-023-04168-7

Leevy, J.L. and Khoshgoftaar, T.M., 2020. A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data. *Journal of Big Data, 7*(1), p.104. https://doi.org/10.1186/s40537-020-00382-x

Long, Z., Yan, H., Shen, G., Zhang, X., He, H. and Cheng, L., 2024. A Transformer-based network intrusion detection approach for cloud security. *Journal of Cloud Computing, 13*(1), p.5. https://doi.org/10.1186/s13677-023-00574-9

Milosevic, M.S. and Ciric, V.M., 2022. Extreme minority class detection in imbalanced data for network intrusion. *Computers & Security, 123*, p.102940. https://doi.org/10.1016/j.cose.2022.102940

Neupane, S., Ables, J., Anderson, W., Mittal, S., Rahimi, S., Banicescu, I. and Seale, M., 2022. Explainable intrusion detection systems (x-ids): A survey of current methods, challenges, and opportunities. *IEEE Access, 10*, pp.112392-112415. 10.1109/ACCESS.2022.3216617

Nzuva, S.M., 2024. A novel bagging-XGBoost ensemble model for attaining high accuracy and computational efficiency in network intrusion detection. *Available at SSRN 4944137*. http://dx.doi.org/10.2139/ssrn.4944137

Ohtani, T., Yamamoto, R. and Ohzahata, S., 2024. IDAC: federated learning-based intrusion detection using autonomously extracted anomalies in IoT. *Sensors, 24*(10), p.3218. https://doi.org/10.3390/s24103218

Pandey, P. and Kapoor, A., 2025. Cybercrime in the Digital Era: Impacts, Awareness, and Strategic Solutions for a Secure Future. *Sachetas, 4*(1), pp.32-37. https://doi.org/10.55955/410004

Saeed, S., Altamimi, S.A., Alkayyal, N.A., Alshehri, E. and Alabbad, D.A., 2023. Digital transformation and cybersecurity challenges for businesses resilience: Issues and recommendations. *Sensors, 23*(15), p.6666. https://doi.org/10.3390/s23156666

Sinha, P., Sahu, D., Prakash, S., Yang, T., Rathore, R.S. and Pandey, V.K., 2025. A high performance hybrid LSTM CNN secure architecture for IoT environments using deep learning. *Scientific Reports, 15*(1), p.9684. https://doi.org/10.1038/s41598-025-94500-5

Srilatha, D. and Thillaiarasu, N., 2024. LSTM-CNN: a deep learning model for network intrusion detection in cloud infrastructures. *International Journal of Critical Infrastructures, 20*(6), pp.505-523. https://doi.org/10.1504/IJCIS.2024.142451