MAJOR PROJECT REPORT

ON

NETWORK TRAFFIC ANOMALY DETECTION USING MACHINE LEARNING.

IN PARTIAL FULFILLMENT OF
B.Tech. (AIML)
[2021-25]

Submitted By:
Akshat Kumar
08419011621
AIML-B2

Guided By:
Dr. Ashish Joshi
Assistant Professor, USAR.

UNIVERSITY SCHOOL OF AUTOMATION AND ROBOTICS, GGSIPU

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY
EAST DELHI CAMPUS, SURAJMAL VIHAR

DECLARATION BY THE CANDIDATE

I hereby declare that the work, which is being presented in this project entitled "Network traffic anomaly detection using Machine Learning", is an authentic record of my own work carried out by me under the supervision and guidance of Dr. Ashish Joshi, Assistant Professor, University School of Automation of Robotics, GGSIPU.

This project was undertaken as a part of the major project report for the partial fulfillment of B.Tech. (Artificial Intelligence and Machine Learning).

I have not submitted the matter embodied here in this project for the award of any other Degree/Diploma

(STUDENT SIGN)

Akshat Kumar,

08419011621,

B.Tech.-AIML.

CERTIFICATE

This is to certify that this MAJOR PROJECT REPORT “Network traffic anomaly detection using Machine Learning” is submitted by

Akshat Kumar, who carried out the project work under my supervision. I approve this project for submission of the Bachelor of Technology (B.Tech., AIML) in the University School of Automation and Robotics (USAR), Guru Gobind Singh Indraprastha University, Delhi.

Date:

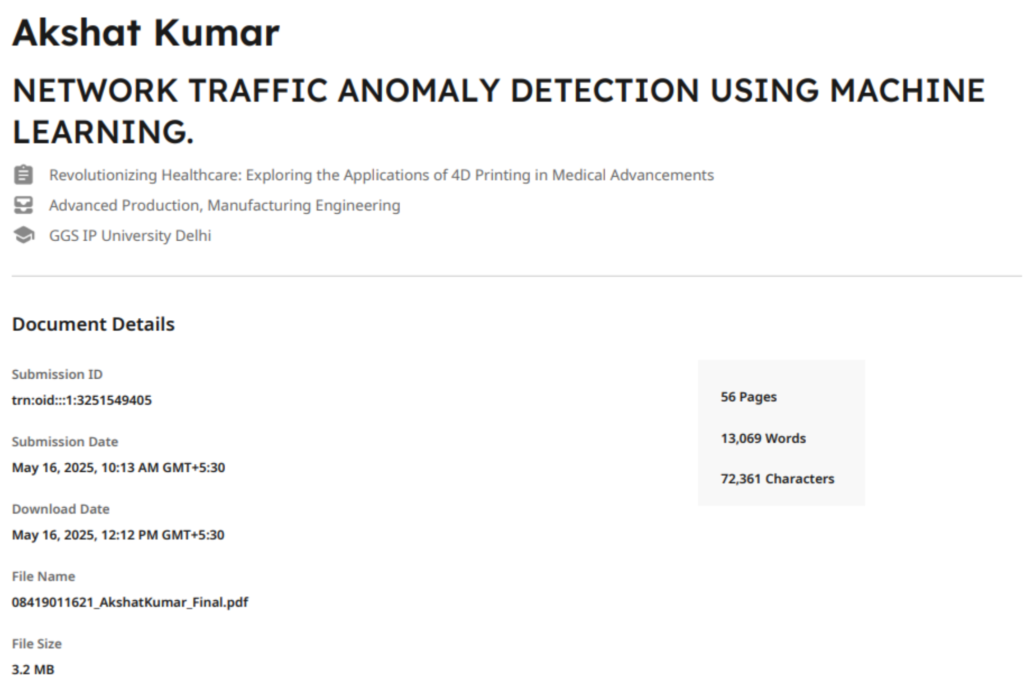
Dr. Ashish Joshi,

Assistant Professor.

USAR, GGSIPU.

PLAGIARISM REPORT

Similarity: 9% Overall Similarity.



Akshat Kumar

NETWORK TRAFFIC ANOMALY DETECTION USING MACHINE LEARNING.

Revolutionizing Healthcare: Exploring the Applications of 4D Printing in Medical Advancements

Advanced Production, Manufacturing Engineering

GGS IP University Delhi

Document Details

Submission ID
trn:oid:::1:3251549405

Submission Date
May 16, 2025, 10:13 AM GMT+5:30

Download Date
May 16, 2025, 12:12 PM GMT+5:30

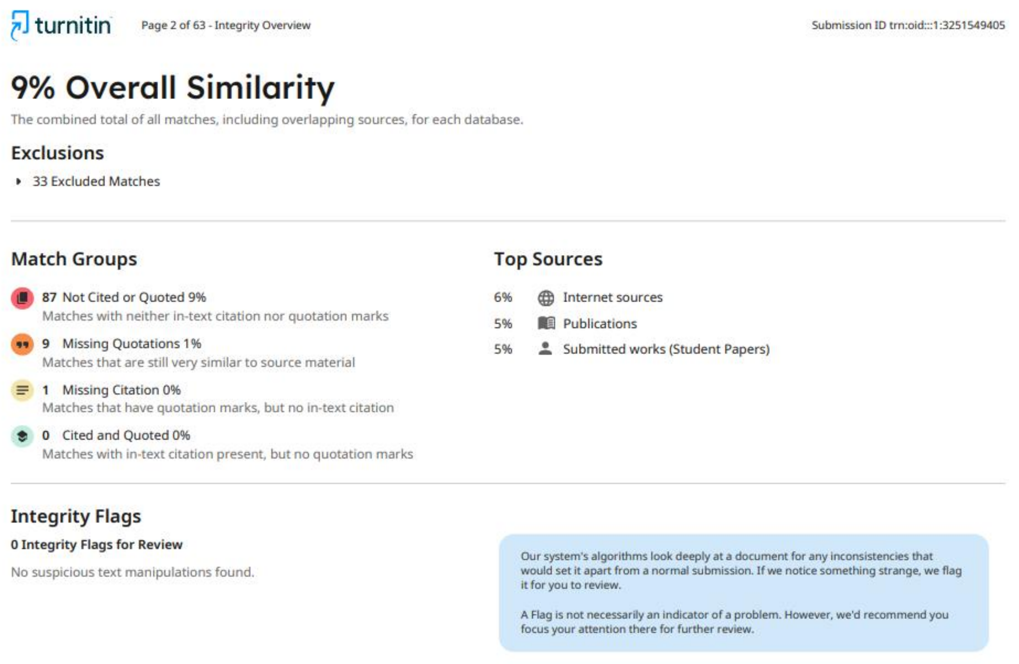
File Name
08419011621_AkshatKumar_Final.pdf

File Size
3.2 MB

56 Pages

13,069 Words

72,361 Characters



turnitin Page 2 of 63 - Integrity Overview Submission ID trn:oid:::1:3251549405

9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Exclusions

- 33 Excluded Matches

Match Groups

- 87 Not Cited or Quoted 9%
  Matches with neither in-text citation nor quotation marks
- 9 Missing Quotations 1%
  Matches that are still very similar to source material
- 1 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

6%	Internet sources
5%	Publications
5%	Submitted works (Student Papers)

Integrity Flags

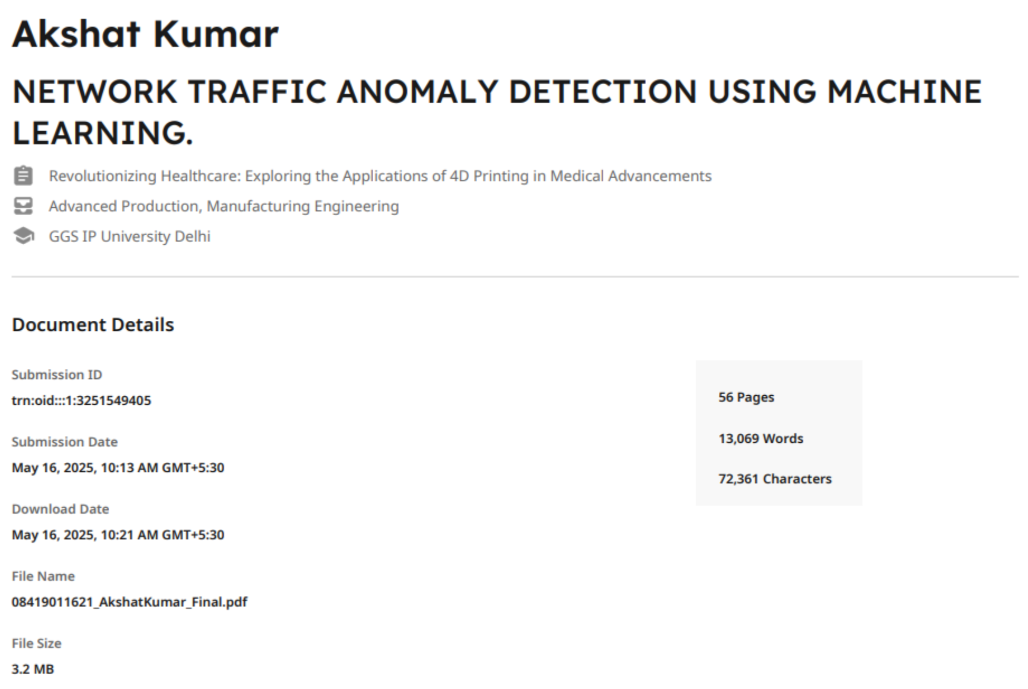
0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

AI Detection: *% Negligible Ai Detected.



Akshat Kumar

NETWORK TRAFFIC ANOMALY DETECTION USING MACHINE LEARNING.

Revolutionizing Healthcare: Exploring the Applications of 4D Printing in Medical Advancements

Advanced Production, Manufacturing Engineering

GGS IP University Delhi

Document Details

Submission ID
trn:oid:::1:3251549405

Submission Date
May 16, 2025, 10:13 AM GMT+5:30

Download Date
May 16, 2025, 10:21 AM GMT+5:30

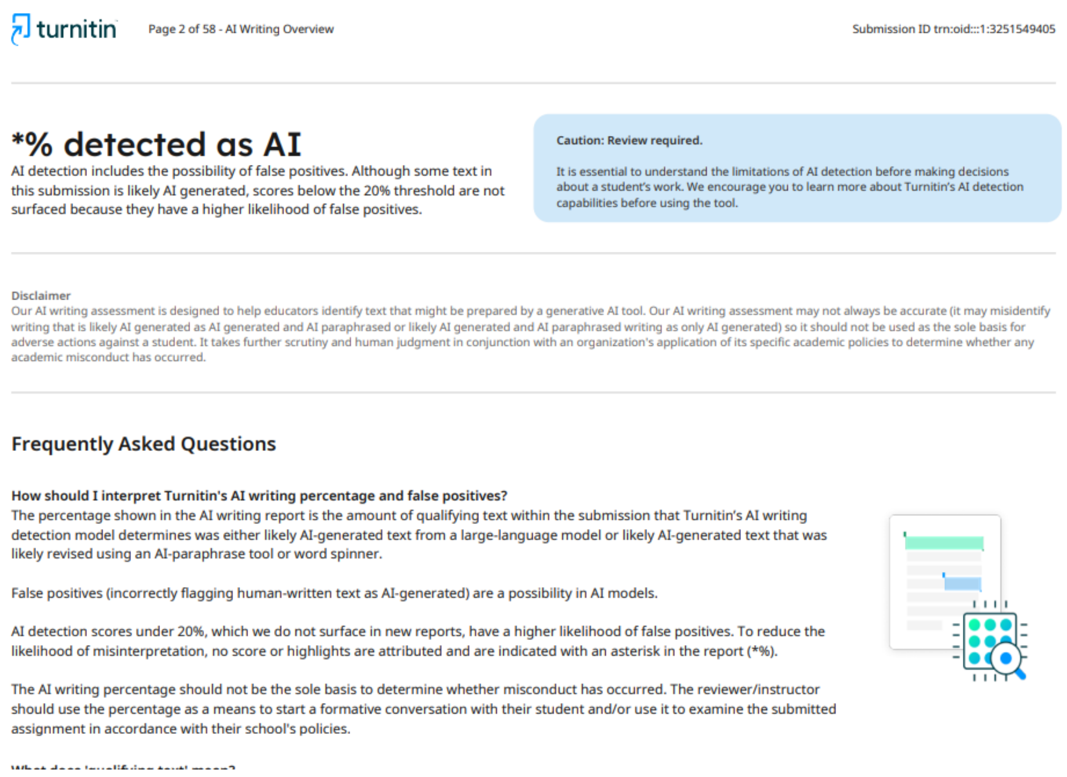
File Name
08419011621_AkshatKumar_Final.pdf

File Size
3.2 MB

56 Pages

13,069 Words

72,361 Characters



turnitin Page 2 of 58 - AI Writing Overview Submission ID trn:oid:::1:3251549405

***% detected as AI**

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have supported and guided me throughout the course of this project. First and foremost, I am deeply thankful to Prof. A. K. Saini, Director, East Campus, GGSIPU, for providing the necessary academic environment and resources that made this project possible. I would also like to extend my heartfelt gratitude to Prof. Arvinder Kaur, Dean USAR, for her constant encouragement and support during this academic journey.

I sincerely thank Prof. Dr. Abha Agarwal, Program Coordinator, for her valuable guidance in structuring the project and ensuring smooth progress. I am especially grateful to Prof. Dr. Manisha Parlewar, Teacher-in-Charge, for the consistent academic supervision, helpful insights, and feedback that significantly enhanced the quality of my work.

A very special thanks to my project mentor, Prof. Dr. Ashish Joshi, for his unwavering support, expert guidance, and constant motivation throughout every phase of this project. Their dedication and mentoring were crucial to the completion of this work.

Lastly, I would like to thank all my faculty members, good friends, and family who contributed in any way toward the success of this project.

Thanking You

Akshat kumar,

08419011621

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

ABSTRACT

It's really quite something when you stop to think about it: every single day, all year round, the internet is buzzing. We're talking about literally thousands upon thousands of us individuals, alongside countless businesses, organizations, and all sorts of entities, constantly interacting in this vast digital space. Over the last few years, the number of accounts using the web has just skyrocketed – it's grown at a truly astounding rate. But with all this incredible connectivity, there's a shadow side: cyberattacks have also become far more frequent, and they're not just simple pranks anymore; they've evolved to become incredibly complex and sophisticated. Often, the straightforward, traditional cybersecurity methods we rely on are designed to defend against known threats. While they have their place, these simpler approaches can sometimes be behind when faced with a brand of unseen, original types of attack or those which seem to come out of nowhere, appearing almost random. This is precisely where a different strategy, known as anomaly-based detection, offers a really promising new path forward, giving us a way to potentially identify these previously unseen and unknown threats.So, with that backdrop, our project really dives headfirst into this challenge. The main goal we set out to achieve was to get better at spotting these "network anomalies" – those unusual blips or strange patterns in network traffic that could signal something fishy is going on – by applying and using the power of various methods as well as techniques. We knew we needed a really good, comprehensive dataset to train and test our ideas, which is why we chose the CICIDS2017 dataset. This particular dataset is fantastic because it's packed with a really broad and diverse range of modern attack types, giving our models a rich learning environment. Before we even let our machine learning models loose, we did some important groundwork: we performed feature selection using a clever technique called Random Forest Regressor. This helped us pinpoint the most important characteristics in the data that could indicate an attack. Once the data was transformed and refined through this process, we then brought in the big guns: seven distinct and varied machine learning models. We didn't just pick similar ones; we chose a range to see how different approaches would fare. And the exciting part? Each of these models actually performed really well, showing strong capabilities in distinguishing between normal traffic and attacks. To give you a clearer picture, here's how their accuracies stacked up, Naive Bayes clocked in at an impressive 86%, Quadratic Discriminant Analysis (QDA) also achieved a solid 86%, Random Forest, a real workhorse in machine learning, reached 94%, The ID3 algorithm did even better, hitting 95%, AdaBoost also performed strongly at 94%, Our MLP, scored 83%, and leading the pack, K-Nearest Neighbors (KNN) demonstrated an outstanding 97% accuracy.

CHAPTER 1: INTRODUCTION:

1.1 **Topic Introduction:**

Today’s highly connected digital environment of today, network integrity must be protected above all. Concurrently, the volume and complexity of cyberattacks have grown exponentially, too. Legacy Intrusion Detection Systems (IDS), which detected attack based on signature-based approaches, are likely to fall short in detecting previously unseen (zero-day) attacks or sinister anomalies. The growing sophistication of cyber threats has sparked a notable shift toward smarter, more responsive security approaches that leverage data analytics, particularly Machine Learning and AI technologies [1,3].

Within this evolving landscape, anomaly detection stands out as a particularly valuable technique. Drawing from unsupervised or semi-supervised learning principles, it identifies network traffic patterns that deviate from established norms, flagging them as potentially harmful. These irregularities (which security professionals sometimes call "outliers" in casual conversation) might indicate anything from unauthorized access attempts to data theft operations. What makes these detection systems truly valuable isn't just their accuracy, but their ability to work effectively across different network environments without bombarding analysts with false alarms.

Our project digs into the practical application of machine learning for anomaly detection using “the CICIDS2017 dataset”[9]—a collection I chose specifically because it captures realistic network traffic combining everyday legitimate activities with various attack scenarios. I've spent countless hours working through the preprocessing challenges (some of which proved surprisingly stubborn!), from basic data cleaning to more sophisticated feature selection. Through this process, we're aiming to identify which models and configurations deliver the most reliable results for real-world threat detection [1,3].

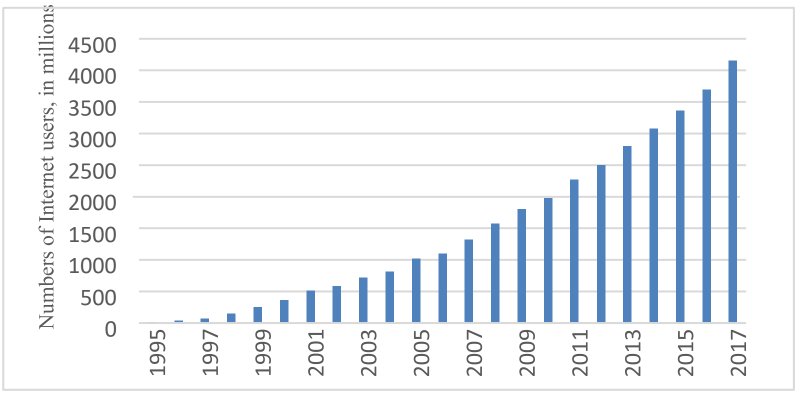


figure 1. Graph showing an increase in the number of users till 2017 [1,2]

1.2 Problem Statement:

The fundamental challenge in network security boils down to one critical factor: can we spot malicious activity quickly and accurately? It's harder than it sounds. Traditional signature-based intrusion detection systems like Snort or Suricata (which I've personally battled with during late-night security implementations) rely on predefined rules and known attack patterns. The problem? They're practically useless against novel threats or those that evolve rapidly.

To make matters worse, we're drowning in data. Network traffic generates massive amounts of high-dimensional information, and—let's face it—the vast majority of this traffic is completely legitimate. Finding that needle in the haystack is no small feat [1]. Several key hurdles make this particularly tricky: First, false positives are killing us. I've seen security teams simply tune out alerts after weeks of traditional systems crying wolf about perfectly normal network behavior. Alert fatigue is real, and it's dangerous. Then there's the generalization problem. Signature-based systems just can't handle zero-day attacks. Scale presents another headache. Networks keep growing (sometimes seemingly overnight), and trying to process everything in real-time quickly becomes computationally expensive. Not all data is created equal, either. Network traffic includes tons of features that, frankly, don't contribute much to detection effectiveness. It's like trying to find your keys while being distracted by everything else in your house. Lastly, the class imbalance issue can't be ignored. Datasets like CICIDS2017 [9] are heavily skewed toward normal traffic, which, without proper handling, can seriously mislead our classification algorithms. In light of these issues, the project looks forward to developing a machine learning-driven approach that can overcome these challenges by learning from historical data and identifying suspicious behaviour based on behavioural anomalies [1,4,5].

1.3 Objective:

The main goal of this project aims to develop, design, and evaluate a network traffic anomaly detection system based on and around machine learning algorithms. The objectives are stated as follows:

1. Preprocess and analyse the CICIDS2017 dataset [9] :

• Handle null/missing values.

• Scale and normalize numeric attributes.

• Label or one-hot encode categorical attributes.

• Remove duplicate records and outliers.

• Remove feature redundancy using correlation thresholds and feature selection.

2. To use Random Forest-based feature importance and RFECV:

•Take the features ranking based on their contribution to classification.

•Support low-impact features for improving the efficiency of the model and to reduce overfitting.

3. To train and compare a variety of classifiers:

•Attempt performance evaluation of various models like Naive Bayes, QDA, Multi-Layer Perceptron, Random Forest, and various other models [6].

•Compare their results using F1-score, precision, ROC-AUC, accuracy, and recall as the evaluation basis.

4. To visualise as well as interpret results:

• Use confusion matrices, classification reports, and accuracy plots to understand model behaviour.

• Evaluate trade-offs between complexity and accuracy.

5. To make inferences and guide the future:

• Identify patterns that indicate good anomaly detection.

• Suggest improvements for applying such a system to real-time environments.

CHAPTER 2: LITERATURE REVIEW:

2.1. Topic Introduction:

The rapid and unstoppable growth of internet-connected devices as well as the corresponding rise in network infrastructure complexity are hallmarks of the digital age. Despite encouraging creativity and communication, this interconnectedness has regrettably resulted in a rise in the quantity, complexity, and possible impact of cyberthreats. The dynamic nature of these threats, especially novel (zero-day) and polymorphic attacks that can circumvent predetermined rules, is too much for traditional network security mechanisms, where they frequently rely heavily upon signature-based detection. This calls for a paradigm change toward defense systems that are more proactive, intelligent, and adaptive. Machine learning (ML)-based Network Anomaly Detection Systems (NADS) [1,6] have become a very promising approach. The goal of these systems is to create a baseline of typical network behavior and then spot statistically significant departures from it, which could be signs of malicious activity, policy infractions, or system issues. With an emphasis on datasets, feature engineering methodologies, the range of model diversity, and noteworthy emerging trends influencing the future of this crucial field, this literature review examines significant research contributions, important trends, and enduring difficulties in the application of various machine learning techniques to the [6] complex problem of network anomaly detection. Finding network anomalies is a complex task. It involves more than just detection; it involves attaining a high level of accuracy in differentiating between real threats and genuinely benign network fluctuations or normal but odd traffic patterns. These threats can be anything from highly advanced advanced persistent threats (APTs)[1,2] and covert data exfiltration attempts to relatively simple but disruptive denial-of-service (DoS) attacks and port scanning. With their innate ability to extract intricate patterns and connections from large, high-dimensional network data, machine learning (ML) algorithms have the potential to more effectively automate the detection of known attack signatures and—more importantly—to spot zero-day anomalies that have not yet been identified by signatures. For contemporary cybersecurity postures, this proactive capability is essential.

2.2. The Foundational Role of Datasets and Feature Engineering:

The ultimate efficacy and reliability of any ML-based NADS [4,5] are inextricably linked to two fundamental pillars: the quality, representativeness, and relevance of the dataset used for training and rigorous evaluation; and the sophistication and appropriateness of the methods employed for feature selection, feature engineering, and dimensionality reduction.

2.3. Evolution of Network Intrusion Datasets:

The historical trajectory of network intrusion detection research has been significantly influenced by the available benchmark datasets. Early research efforts frequently utilized datasets such as KDD99 and its refined successor, NSL-KDD. For instance, **Revathi and Malathi (2013)** [10] employed the KDD99 dataset to conduct a comparative analysis of various ML algorithms, including decision trees, SVMs, and Naive Bayes, where they particularly emphasized the critical importance of comprehensive data preprocessing steps for achieving meaningful results. Similarly, **Kayode-Ajala (2021)** [11] utilized the NSL-KDD dataset to evaluate the performance of seven distinct ML models, with their findings highlighting the strong performance of K-Nearest Neighbors (KNN) in classifying network traffic. However, these older datasets, while foundational, have faced increasing criticism over time. Their primary limitations include a lack of diversity in attack types, the presence of redundant records, an unrealistic distribution of traffic, and a general failure to fully represent the complexities and characteristics of modern network traffic and sophisticated attack vectors.

Recognizing these significant limitations, the research community has actively worked towards developing newer, more robust datasets. **Moustafa and Slay (2015)** introduced the UNSW-NB15 dataset [12], meticulously designed to offer a more comprehensive, contemporary, and realistic representation of network traffic. This dataset encompasses 49 distinct features across a multitude of modern attack types, thereby providing a more challenging and relevant benchmark. The CICIDS2017 dataset stands as another widely adopted modern dataset, particularly noted for its broad range of contemporary attack types and its inclusion of benign traffic generated through realistic user profiles. **Kostas (2018)** [8], in his master's thesis, specifically leveraged the CICIDS2017 dataset [9] to conduct an in-depth exploration of ML-based anomaly detection. His work involved comparing a suite of models, including Quadratic Discriminant Analysis (QDA), Random Forest , Naive Bayes, Multi-Layer Perceptron (MLP) and KNN [8]. The selection of CICIDS2017 [9] aligns well with projects that aim to develop detection capabilities against a diverse array of modern and evolving network threats, reflecting current cybersecurity challenges[1,2].

2.4. Dimensionality Reduction and Feature Selection:

Network datasets are often characterized by high dimensionality, meaning they contain a large number of features or attributes for each network connection or packet. While rich in information, this high dimensionality can lead to several practical and theoretical problems, collectively known as the "curse of dimensionality." These problems include increased computational costs for model training and inference, a higher chances of model overfitting (where the model learns to take in consideration the noise instead of the signal actually being sent), and the potential for multicollinearity, which can obscure the true importance of individual features. Effective dimensionality reduction and astute feature selection are, therefore, not merely beneficial but crucial steps in the ML pipeline for NADS.

Kayode-Ajala (2021) [11] provided a practical demonstration of the utility of Principal Component Analysis (PCA), a popular unsupervised linear transformation technique. By applying PCA, the NSL-KDD dataset was reduced from 122 features to a more manageable set of 20 important components, which capture the maximum change and variance as well as verity in the data. This substantial reduction in dimensionality significantly lowered computational costs without leading to a major detrimental result on the classification performance as well as utility of the K-Neighbors classifier, which still achieved high accuracy. The work by **Revathi and Malathi (2013)** [10] also underscored the synergistic importance of feature selection in conjunction with thorough data preprocessing for achieving effective anomaly detection on the older KDD99 dataset. Further reinforcing this, **Kostas (2018)** [8] emphasized that feature selection, particularly employing Random Forest-based methods (which can rank features by their importance in discriminating classes), was a crucial determinant in achieving optimal model performance when working with the CICIDS2017 dataset. These studies collectively and emphatically affirm that thoughtful, data-driven feature engineering and dimensionality reduction are critical precursors to the successful application and deployment of ML models in network security.

2.5. Machine Learning Paradigms for Anomaly Detection

A diverse and expanding array of ML algorithms has been rigorously applied and adapted to the domain of network anomaly detection. These approaches can be broadly categorized, spanning supervised learning (requiring labelled data), unsupervised learning (operating on unlabelled data), and semi-supervised learning (utilizing a mix of labelled and unlabelled data) [6,3].

2.5.1. Traditional Machine Learning Models:

Several foundational studies have focused on benchmarking and comparing the performance of traditional, often statistically based, ML classifiers. As previously mentioned, **Kayode-Ajala (2021)** [11] identified KNN as a highly effective model on the NSL-KDD dataset. KNN operates on the principle of proximity, classifying a data point based on the majority class of its 'k' nearest neighbours in the feature space, making it intuitively suitable for identifying instances that are "far" from normal clusters. **Revathi and Malathi (2013)** [10] provided a comparative analysis of decision trees (which create a tree-like model of decisions), Support Vector Machines (SVMs, which find an optimal hyperplane to separate classes), and Naive Bayes (a probabilistic classifier based on Bayes' theorem with strong independence assumptions). The extensive work by **Kostas (2018)** [8] using the CICIDS2017 [9] dataset offered a valuable comparative evaluation of Naive Bayes, QDA (a probabilistic classifier assuming Gaussian distributions), MLP (a type of feedforward neural network), KNN, and Random Forest. Random Forest, an ensemble learning method that constructs multiple decision trees and outputs the mode of their predictions, emerged as a particularly strong performer in his study. The consistent demonstration of good performance by models like KNN and Random Forest across different datasets and research contexts highlights their inherent robustness and suitability for the complex task of network anomaly classification [3].

2.5.2. Deep Learning Approaches:

In recent years, deep learning, a subfield of ML characterized by neural networks with multiple layers (deep architectures), has garnered significant attention and achieved remarkable successes across various domains, including network anomaly detection. Its primary allure lies in its ability to automatically learn hierarchical feature representations directly from raw or minimally processed complex data, thereby reducing the need for manual, time-consuming feature engineering. **Chalapathy and Chawla (2019)** [13] provide a comprehensive and insightful survey of various deep learning techniques specifically tailored for anomaly detection. They categorize these methods into supervised, semi-supervised, and unsupervised approaches, discussing the architectures, strengths, and weaknesses of models such as autoencoders (neural networks trained to reconstruct their input, with anomalies often having higher reconstruction errors), Convolutional Neural Networks (CNNs, particularly effective for spatial hierarchies in data, though less directly applicable to tabular network data unless transformed), Recurrent Neural Networks (RNNs, including LSTMs and GRUs, which are well-suited for sequential data like network traffic flows), and Generative Adversarial Networks (GANs, which can learn the distribution of normal data and identify outliers). While these deep learning models are undeniably powerful and capable of capturing intricate patterns, the authors prudently caution that they often come with significant prerequisites: large, high-quality labeled datasets for supervised training, and substantial computational resources (e.g., GPUs) for both training and, in some cases, real-time inference.**Yin et al. (2017)** [14] specifically explored the application of Deep Belief Networks (DBNs), a type of generative graphical model composed of multiple layers of stochastic latent variables, for the task of feature learning directly from raw network data. Their approach, when evaluated on the NSL-KDD dataset [10] , reportedly outperformed traditional ML models, thereby demonstrating the distinct advantage of deep learning in autonomously extracting complex and often subtle patterns that might be missed by conventional methods or require extensive manual feature engineering.

2.5.3. Unsupervised Learning Methods:

Unsupervised learning methods hold particular value and promise in the context of network anomaly detection, primarily because they do not require pre-labeled data. This makes them exceptionally well-suited for detecting novel, zero-day anomalies for which signatures or labels are, by definition, unavailable. **Ramadas, Ostermann, and Tjaden (2003)** [15] , in an early but influential work, employed Self-Organizing Maps (SOMs), a type of unsupervised neural network that uses competitive learning to produce a low-dimensional, discretized representation (a map) of the input space of the training samples. They used SOMs to visualize network traffic patterns and thereby detect anomalies as deviations from learned normal clusters. This approach, and others like clustering (e.g., k-means, DBSCAN) or density estimation, offers a viable and often necessary alternative for identifying previously unknown threats, a critical capability for maintaining robust security in constantly evolving threat landscapes.

2.6. Advanced Concepts and Comprehensive Surveys

Beyond the application of individual model types, the research field has also actively explored more sophisticated data representations, the crucial aspect of interpretability (or explainability) of detection systems, and the synthesis of knowledge through broad surveys.

2.6.1. Graph-Based Anomaly Detection:

Network traffic, with its inherent relational structure, can often be naturally and powerfully represented as a graph. In such representations, nodes can symbolize entities (e.g., IP addresses, users, devices), and edges can represent interactions or connections between them (e.g., data flows, communication sessions). **Akoglu, Tong, and Koutra (2014)** [16] provided a foundational survey of anomaly detection techniques specifically designed for graph data. Their work covered various types of graphs, including static and dynamic graphs, as well as attributed (nodes/edges have features) and plain graphs. They particularly highlighted the importance of leveraging structural features of the graph (e.g., node degrees, clustering coefficients, community structures, path lengths) for effective anomaly detection. Building upon this, **Ma et al. (2021)** [2] examined the application of deep learning approaches to graph anomaly detection, detailing methods for identifying node-level, edge-level, subgraph-level, and even entire graph-level anomalies. Their survey also outlined available open datasets for graph-based detection, relevant evaluation metrics, and promising future research avenues. These works collectively suggest that adopting graph representations can offer richer, more contextual insights into network behavior and can uncover anomalies that might be obscured in traditional tabular data formats. For instance, a sudden change in the connectivity pattern of a critical server or the appearance of an unusually isolated cluster of nodes could be indicative of an anomaly.

2.6.2. Explainability in Anomaly Detection:

As ML models, particularly deep learning architectures, become increasingly complex and "black-box" in nature, the ability to understand *why* a particular instance (e.g., a network connection) is flagged as an anomaly becomes critically important. This is especially true for security analysts in Security Operations Centers (SOCs) who need to validate alerts, understand the nature of a potential threat, and decide on appropriate response actions. **Nguyen et al. (2019)** [17] directly addressed this challenge by proposing GEE (Gradient-Based Explainable Variational Autoencoder). Their method innovatively combines Variational Autoencoders (VAEs), a type of generative model, with gradient-based explanation techniques (which highlight input features most responsible for a given output). Evaluated on the UGR'16 dataset, GEE demonstrated effective results in both detecting and providing human-understandable explanations for network anomalies. Such research promotes the development of interpretable and unsupervised learning models for network security, fostering trust and facilitating more effective human-machine collaboration in threat response [3].

2.6.3. Broad Surveys and Identified Challenges:

Several comprehensive surveys offer a wider, panoramic lens on the entire field of ML for intrusion detection, synthesizing findings and identifying overarching themes. **Buczek and Guven (2016)** [19] reviewed an extensive range of ML techniques applicable to intrusion detection, encompassing supervised, unsupervised, and semi-supervised methods. Their work was particularly valuable in emphasizing persistent and significant challenges that the field continues to grapple with. These include dataset skewness (the inherent imbalance where anomalous traffic is far less frequent than normal traffic, which can bias model training), the critical need for real-time detection capabilities to handle high-velocity network streams, and the difficulty of evaluating models under realistic operational conditions. These challenges directly resonate with the limitations also noted by **Chalapathy and Chawla (2019)** [18] regarding the often substantial data and computational resource demands of advanced deep learning models.

2.7. Synthesis, Challenges, and Relevance to Current Endeavours:

The reviewed body of literature paints a vibrant, dynamic, and rapidly evolving picture of the network anomaly detection field. There is a discernible and ongoing trend towards leveraging more sophisticated, realistic, and contemporary datasets like UNSW-NB15 and CICIDS2017 [9], as these better reflect the complexities of modern network environments and attack strategies. Feature selection and dimensionality reduction remain pivotal, indispensable steps in the ML pipeline, with methods like PCA for unsupervised reduction and Random Forest-based selection for supervised or importance-guided reduction proving consistently effective in enhancing model performance and efficiency.

While traditional ML models such as KNN and Random Forest continue to demonstrate strong and reliable performance across various benchmarks, the allure of deep learning techniques is undeniably growing [6]. This is primarily due to their remarkable capacity for automatic feature learning from raw data, thereby potentially reducing the burden of manual feature engineering. However, this power comes with important caveats, primarily concerning the need for large, well-curated datasets and significant computational overhead. Unsupervised methods continue to hold significant promise, especially for the crucial task of zero-day attack detection, where prior knowledge of attack signatures is absent. Furthermore, the exploration of advanced areas like graph-based analytics and explainable AI (XAI) points towards exciting future advancements in creating NADS that are not only more accurate but also more insightful, transparent, and trustworthy for human operators.

Despite these advancements, key challenges persist and demand ongoing research focus:

- **Dataset Quality, Availability, and Representativeness:** The scarcity of large, diverse, publicly available, and accurately labeled datasets remains a significant hurdle, especially for training robust supervised deep learning models. The cost and effort involved in generating and labeling such datasets are substantial. Moreover, ensuring datasets remain representative of evolving network traffic and attack vectors is an ongoing concern.

- **Computational Resources and Scalability:** Complex models, particularly deep learning architectures with many layers and parameters, demand considerable computational power (CPUs, GPUs, TPUs) for training. Moreover, for practical deployment, inference must often occur in real-time or near real-time, which can also be resource-intensive, especially at scale for high-throughput networks.

- **Real-Time Detection and Latency:** Many academic studies and proof-of-concept systems operate in an offline, batch processing mode. Transitioning these research prototypes to effective real-time detection systems capable of handling the high speed and volume of modern network traffic without introducing unacceptable latency is a non-trivial engineering and algorithmic challenge.

- **Dataset Skewness/Imbalance and Rare Event Detection:** Anomalies are, by their very definition, rare events compared to the vast amount of normal network traffic. This inherent class imbalance can significantly bias ML models during training, leading them to perform well on the majority (normal) class but poorly on the minority (anomalous) classes, resulting in high false negative rates for critical threats. Techniques like oversampling, undersampling, synthetic data generation (e.g., SMOTE), and cost-sensitive learning are often employed to mitigate this [1].

- **Adaptability, Concept Drift, and Model Maintenance:** Network traffic patterns are not static; they evolve due to changes in user behavior, new applications, and infrastructure updates. Similarly, attack methodologies are constantly changing. This phenomenon, known as concept drift, can degrade the performance of a trained NADS over time. Systems must therefore incorporate mechanisms for continuous learning, model retraining, or adaptation to maintain their efficacy.

For projects that are specifically focused on detecting network anomalies using contemporary datasets like CICIDS2017 and are employing a diverse range of ML models (such as Naive Bayes, QDA, Random Forest, ID3, AdaBoost, MLP, KNN) [6,1] subsequent to performing rigorous feature selection (for example, using Random Forest Regressor to identify influential features), the existing body of literature provides strong validation and a solid foundation. The high accuracy figures reported for models such as KNN (often achieving upwards of 97%), ID3 (around 95%), Random Forest (around 94%), and AdaBoost (also around 94%) in such contexts are consistent with findings where these classifiers, or the ensemble/probabilistic principles they are based on, have demonstrated robust and reliable performance (e.g., as seen in the work of **Kayode-Ajala, 2021** [11] for KNN; and **Kostas, 2018** [8] for Random Forest). The systematic and methodical approach involving careful feature selection followed by a comprehensive comparative model evaluation is a well-established and respected methodology within this research domain.

2.8. Conclusion

Machine learning offers an increasingly powerful and indispensable toolkit for significantly enhancing network security through the intelligent detection of anomalous activities. The research landscape in this area is exceptionally rich and multifaceted, featuring a wide spectrum of diverse approaches, ranging from well-established statistical models and traditional classifiers to cutting-edge deep learning architectures and sophisticated graph-based analytical techniques. While substantial progress has undeniably been made in improving detection rates and reducing false alarms, ongoing research continues to vigorously address the persistent challenges related to data quality and availability, computational efficiency and scalability, real-time applicability and low-latency processing, robust handling of imbalanced data, and the crucial aspect of model interpretability and explainability.

Future work in network anomaly detection will likely focus on several promising directions. These include the development of more sophisticated hybrid models that combine the strengths of different algorithmic approaches (e.g., deep learning for feature extraction coupled with traditional classifiers for decision making), advancements in unsupervised and semi-supervised learning techniques to better tackle zero-day threats and reduce labeling efforts, the deeper integration of domain-specific knowledge into ML models, and the development of more robust frameworks for continual learning and adaptation to concept drift. The overarching pursuit of ever-higher accuracy and improved detection capabilities, when coupled with critical practical considerations such as explainability, resource efficiency, and ease of deployment, will undoubtedly drive the innovation and evolution of the next generation of Network Anomaly Detection Systems, making our digital interactions safer and more secure.

CHAPTER 3: METHODOLOGY:

3.1 Objective:

At the base of what we're trying to achieve with this project is something pretty ambitious but incredibly important: we want to build a really smart and adaptable system – a complete framework, if you will – that can sniff out cyber threats in network traffic. We're not just talking about the usual suspects, the kinds of attacks that security systems already know about. The real challenge, and where we're focusing a lot of our energy, is in equipping this framework with advanced machine learning smarts so it can also spot brand new, never-before-seen threats – those "unknown unknowns" that can cause so much damage.

Think of it like a highly skilled digital detective. Its main job will be to constantly watch the flow of data on a network and make a crucial distinction: is this traffic friendly and normal, or is there something malicious lurking within? And because cyberattacks can happen in a heartbeat, it’s vital that our system can do this analysis incredibly quickly, almost as it's happening – what we call near real-time. To make sure our detective is learning from the best and most relevant information, we're using the CICIDS2017 dataset [9], specifically looking at the streams of features it provides, which gives us a rich picture of network behaviour.

Now, how do we plan to teach our system to be so discerning? We're not putting all our eggs in one basket. We'll be diving deep into both major families of machine learning. On one hand, we'll use supervised methods, where we essentially show the system examples of "good" and "bad" traffic so it can learn the difference. On the other hand, and this is key for finding those novel threats, we'll be exploring unsupervised methods, where the system has to figure out what's normal on its own and then flag anything that deviates too much from that baseline.

Before we even let our machine learning models loose on the data, there's a crucial preparation phase – like a chef meticulously preparing ingredients before cooking. This involves a series of pre-processing steps. We'll be carefully normalizing the data (to make sure different scales don't skew the results), encoding it (turning categorical information into numbers our models can understand), and then smartly reducing the number of dimensions or features we're looking at. This isn't about throwing away information; it's about selecting the most important signals and making the learning process more efficient and effective [1,2].

Ultimately, our aim isn't just to say our system "works." We want to rigorously prove its mettle. We'll be looking at a whole suite of performance measures – not just overall accuracy, but also how precise it is (how many of its "bad" flags are actually bad), its recall (how many of the actual threats it manages to catch), and that F1-score, which is applied to balances both recall as well as precision. By optimizing for all these different metrics, we're striving for a truly robust and reliable system that doesn't just perform well on paper but genuinely improves our ability to fight against the ever-evolving field of cyberattacks and threats. It's about building a framework that's not only powerful but even flexible and strong enough to adapt as new challenges emerge.

3.2 Proposed Design Flow:

The methodology of this project is divided into multiple clearly defined stages:

1. Dataset Acquisition
2. Data Preprocessing and Cleaning.
3. Exploratory Data Analysis (EDA)
4. Feature Selection and Engineering
5. Model Selection as well as Hyperparameter Tuning
6. Training and Validation
7. Performance Evaluation and Visualization
8. Comparative Analysis

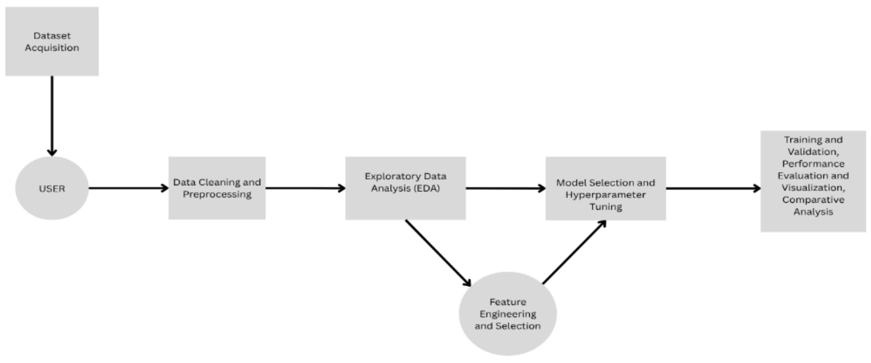
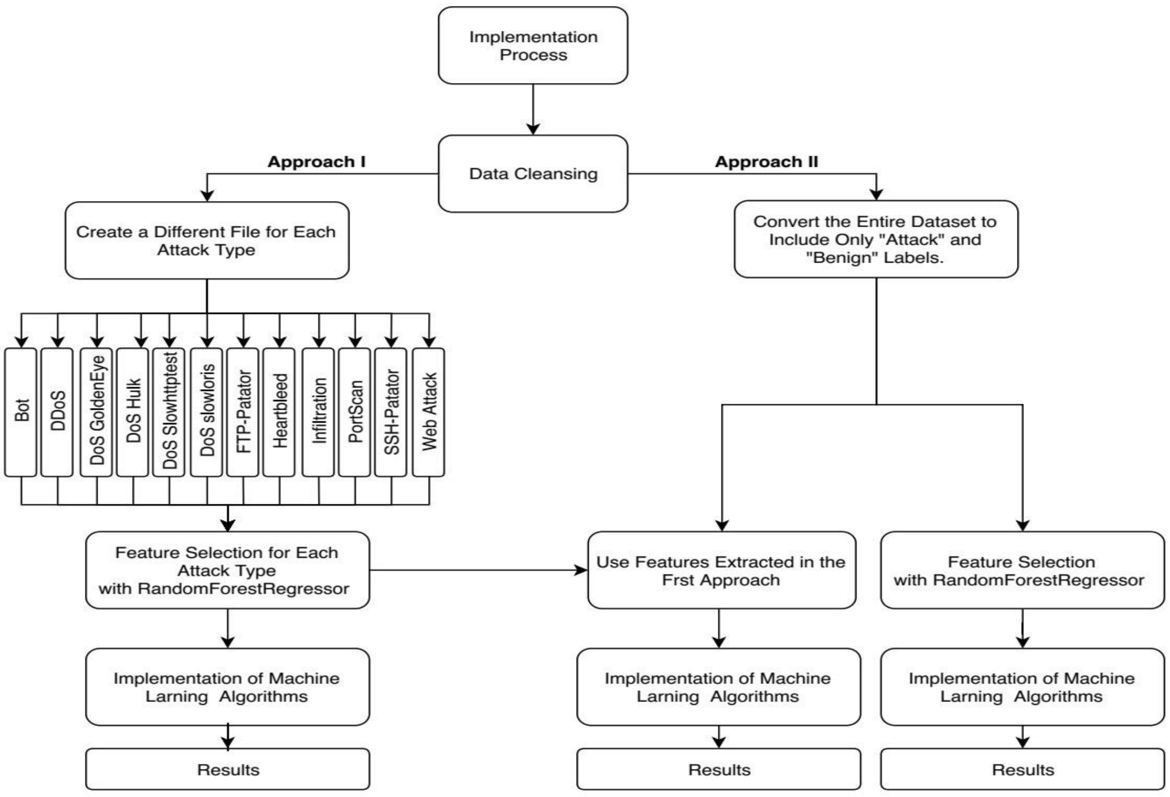

Dataset Acquisition
USER
Data Cleaning and Preprocessing
Exploratory Data Analysis (EDA)
Model Selection and Hyperparameter Tuning
Training and Validation, Performance Evaluation and Visualization, Comparative Analysis
Feature Engineering and Selection

figure 2. Design Flow used.

3.2.1 Implementation Flowchart Design:

A flowchart represents the order and place of operations involved in the anomaly detection process:



Implementation Process
Data Cleansing
Approach I
Approach II
Create a Different File for Each Attack Type
Convert the Entire Dataset to Include Only "Attack" and "Benign" Labels.
Bot
DDoS
DoS GoldenEye
DoS Hulk
DoS Slowhttptest
DoS slowloris
FTP-Patator
Heartbleed
Infiltration
PortScan
SSH-Patator
Web Attack
Feature Selection for Each Attack Type with RandomForestRegressor
Use Features Extracted in the Frst Approach
Feature Selection with RandomForestRegressor
Implementation of Machine Larning Algorithms
Implementation of Machine Larning Algorithms
Implementation of Machine Larning Algorithms
Results
Results
Results

figure 3. The Implementation Process.

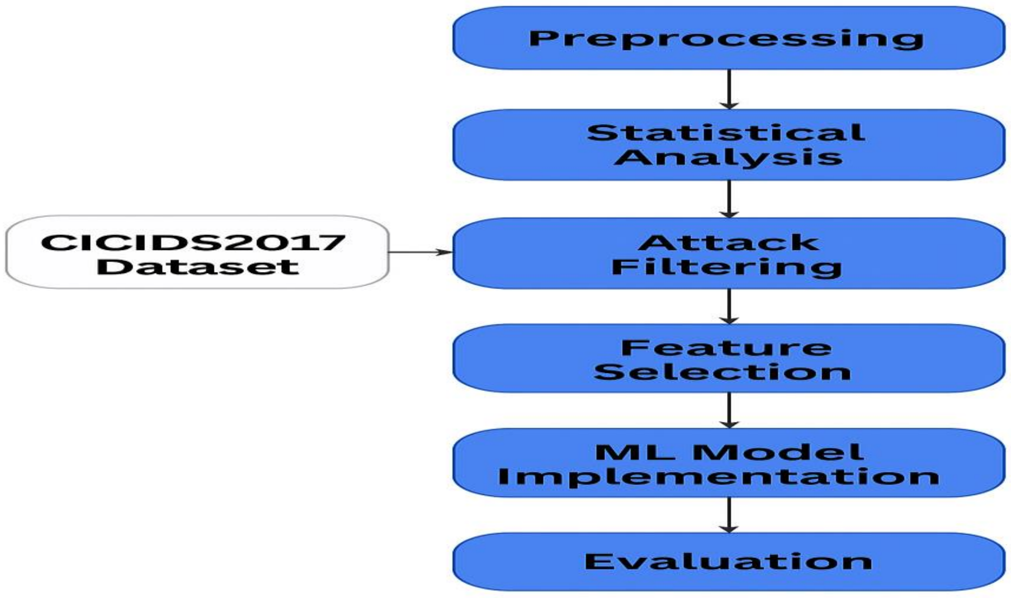


figure 4. The Flow Design Process

3.2.2 Data Flow Diagram (DFD):

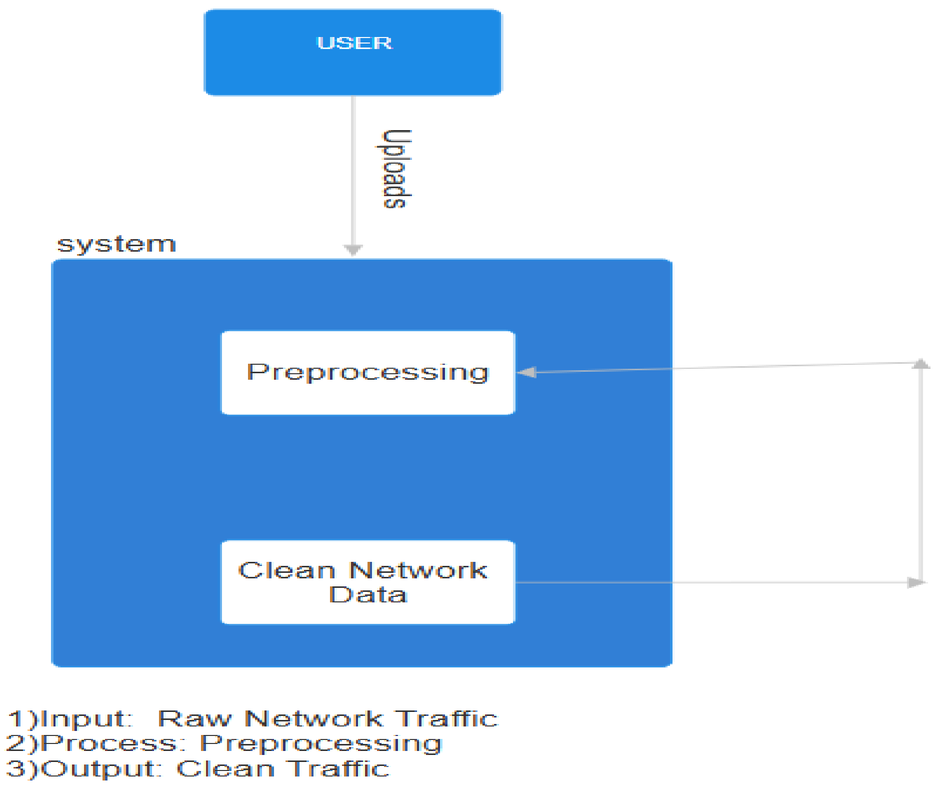


figure 5. Design Flow Diagram.

3.2.3 E-R Diagram:

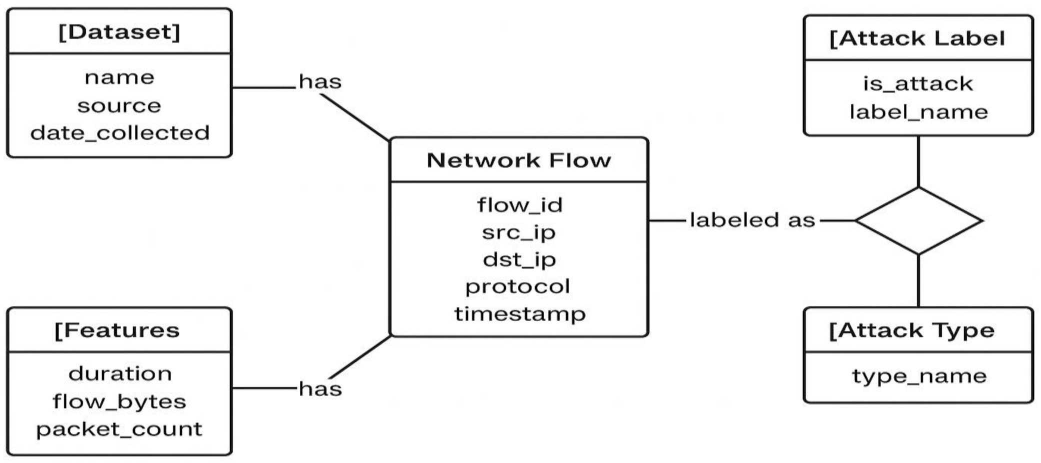


figure 6. Entity-Relationship in the project.

3.3 Proposed Methodology:

3.3.1 Data Acquisition:

The CICIDS2017 dataset [9], provided by the Canadian Institute for Cybersecurity, contains realistic traffic capturing normal and attack behavior in a simulated network environment. This includes brute-force attacks, DoS, botnets, infiltration, and web-based threats [9].

Flow Recording Day (Working Hours)	**Duration**	**CSV File Size (MB)**	**Attack Name**	**Flow Count**	**PCAP file size in GB**
Monday	All Day	257	No Attack	529918	10
Tuesday	All Day	166	FTP-Patator, SSH-Patator [8]	445909	10
Wednesday	All Day	272	DoS Hulk, DoS GoldenEye, DoS Slowloris, DoS Slowhttptest, Heartbleed [8]	692703	12
Thursday	Morning And Afternoon	87.7	Web Attacks (Brute Force, XSS, SQL Injection)	170366	7.7
		103	Infiltration	288602	
Friday	Morning, Afternoon, Afternoon	71.8	Bot	192033	8.2
		92.7	DDoS	225745	
		97.1	PortScan	286467	

table 1: Details of the selected Dataset.

3.3.2 Data Preprocessing and Statistics:

We have a collection of raw data files, specifically several CSV files from the CICIDS2017 dataset, all neatly tucked away in a " CSVs " folder. Our Python script is designed to roll up its sleeves and dive into this data, acting like a meticulous data janitor and then a keen-eyed analyst.

Part One: The Great Data Cleanup and Unification

The first major job of this script is to automate the often tedious but crucial process of cleaning and preparing this data. Raw network traffic data, especially from a comprehensive dataset like CICIDS2017, which captures a wide array of network activities, can be a bit messy. So, the script systematically goes through each CSV file, one by one, and performs a series of important cleansing operations:

1. **Dealing with Imperfections:** It starts by looking for any rows that are either completely empty or have become corrupted, perhaps during data collection or transfer. These incomplete or damaged rows could throw off our analysis later, so the script carefully removes them, ensuring we're only working with sound data.

2. **Standardizing Characters:** Sometimes, text data can contain non-standard characters, like different types of dashes (e.g., "en dashes" instead of standard hyphens). The script diligently standardizes these, converting them to a consistent format. This might seem like a small detail, but it's vital for ensuring that text-based features are treated uniformly.

3. **Handling Missing or Problematic Values:** The script then tackles a common issue in datasets: what to do with blank cells, values that represent infinity (which can crop up in calculations), or entries that are supposed to be numbers but aren't (non-numeric values). Instead of just deleting these or making assumptions, the script replaces them with specific "non-benchmarked placeholders." This means we're acknowledging that a value was problematic without letting it interfere with numerical calculations. It also thoughtfully "zeros out" any placeholders that might have been used inappropriately, ensuring consistency.

4. **Encoding Categorical Data:** Many machine learning algorithms prefer to work with numbers rather than text. So, for all the columns that contain descriptive categories (like types of protocols or services), the script assigns unique numerical codes. Think of it as creating a secret codebook for the data. However, there's one very important exception: the "Label" column. This column tells us whether a particular piece of traffic was normal (benign) or part of an attack, and it's preserved in its original, human-readable form because it's the very thing we're trying to understand and predict.

5. **Shedding Redundancy:** Datasets sometimes contain columns that are either duplicates or don't add any new information. The script identifies and removes one such redundant column, streamlining the dataset and making it more efficient to work with.

Once each individual CSV file has been meticulously processed and polished through these steps, the script then performs a grand unification: it merges all these cleaned-up files into a single, comprehensive master file named all_data.csv. This consolidated file is now ready for the next stage of analysis [8].

Part Two: Unveiling the Story in the Labels

With the all_data.csv file prepared, the script transitions into its analytical role. The focus here is squarely on the "Label" column, which, as we mentioned, holds the key to understanding the different types of network traffic.

1. **Extracting Key Information:** First, it carefully extracts this "Label" column from the massive merged dataset.
2. **Statistical Deep Dive:** The script then performs a statistical analysis on these labels. It's particularly interested in various types of attacks available in the selected dataset and how frequently each one occurs.
3. **Grouping by Frequency:** To make sense of the attack landscape, it groups the various attack labels based on their frequency of occurrence. It categorizes them into three intuitive groups: "high-occurrence" attacks (the ones that appear very often), "medium-occurrence" attacks, and "low-occurrence" attacks (the rarer types).
4. **Visualizing the Findings:** Numbers are great, but pictures often tell a clearer story. So, the script generates bar graphs to visually represent these label distributions. You'll see charts showing how many instances belong to each attack frequency group.
5. **The Bigger Picture: Benign vs. Attack:** Alongside the detailed attack breakdown, the script also calculates and displays the total ratio of benign (normal) traffic compared to all attack traffic. This gives a high-level overview of the dataset's class distribution.

Why go to all this trouble in the second part? Understanding this class distribution – how many examples we have of normal traffic versus different types of attacks, and how common or rare each attack type is – is absolutely fundamental for any project aiming to build an anomaly detection or traffic classification system. It helps in understanding potential biases in the dataset and in designing strategies for training machine learning models effectively [8].

3.3.3 Data attack filter :

The Python script here reads the file `all_data.csv` as input and outputs individual CSV files for each kind of cyberattack available in the dataset. It aggregates all associated attack records and randomly samples benign records at a 30:70 ratio for each kind of attack. It saves these to new files inside an `attacks` subdirectory, which will be created if it does not already exist. Following the addition of the individual attack files, the script combines all "Web Attack" categories – XSS, Brute Force, and SQL Injection – into a single joined file named `web Attack.csv`, then deletes the original individual files. This helps to generate balanced datasets for every attack type, which are best suited for machine learning and analysis.

3.3.4 Feature Importance Extraction:

Now that our data is clean, unified, and we have a good grasp of the different types of activities captured within it, the script embarks on a particularly insightful mission: to figure out which characteristics – or "features" – of the network traffic are the most powerful clues for distinguishing malicious attacks from normal, everyday activity.

This is like asking, "What are the tell-tale signs of trouble?" To answer this, the script employs a sophisticated machine learning technique called Random Forest Regressor. Think of this as assembling a team of expert digital detectives, each looking at the data from a slightly different angle, and then combining their wisdom to make a final judgment.

This part of the script is quite flexible. It's smart enough to analyze data that might have already been separated into different CSV files, perhaps one for each specific type of attack, all residing in a directory conveniently named "attacks". Alternatively, if we want a broader view, it can just as easily process the entire, consolidated all_data.csv file we created earlier. This adaptability allows us to either zoom in on the unique fingerprints of individual attack types or get a holistic view of what matters most across all kinds of threats.

1. **Setting the Stage for Visual Insights:** Before diving into the number-crunching, the script is a good housekeeper. It makes sure a special folder, which we'll call feature_pics (short for "feature pictures"), exists. If it doesn't, the script creates it. This folder will serve as a gallery, storing visual charts that beautifully illustrate which features are the most important.

2. **Loading the Clues and Fine-Tuning the Data:** The script then begins its core work, iterating through each relevant CSV file (whether it's a specific attack file or the main all_data.csv). For each file, it doesn't just blindly grab everything; it selectively loads only the columns (features) that we've previously identified as potentially important, listed in a configuration called main_labels. Once this targeted data is loaded, a bit more pre-processing magic happens. Any lingering missing values that might have slipped through are now filled in with zero – a practical choice to ensure our machine learning models don't stumble. All the numerical data is then converted to a specific format called float32, which is often more memory-efficient for machine learning tasks. And critically, if any "infinite" values are present (which can sometimes occur in network measurements and can cause mathematical headaches for algorithms), the script wisely replaces them with zero. This ensures our analysis is smooth and the results are trustworthy.

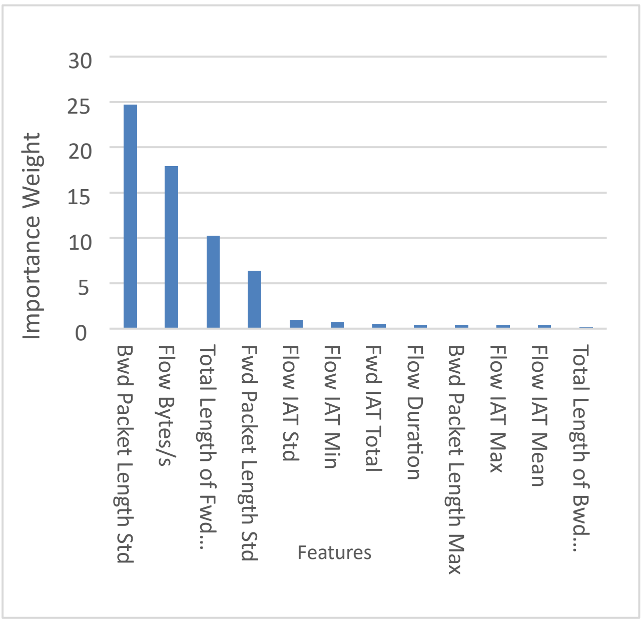
3. **Simplifying the Question: Attack or Not?** To make the Random Forest Regressor's job clearer and more focused, the script reframes the complex world of various attacks into a straightforward "yes/no" question. It prepares the data for what's known as binary classification. This involves a relabeling process: every record that represents any kind of attack is given the label '0' (let's think of this as the "uh-oh, something's wrong" signal), while all the benign, normal traffic records are labeled with '1' (the "all clear" signal). This creates a very clear target for our machine learning model to learn. This binary target variable (often called 'y' in machine learning lingo – it's the answer we want to predict) is then neatly separated from the set of features (often called 'X' – these are the clues the model uses to make its prediction).

4. **Unleashing the Power of the Random Forest:** With the data perfectly prepped, the RandomForestRegressor model is set to work. As mentioned, it isn't only one choice-maker; it's an ensemble, like a committee of many different decision trees.Where Each tree looks at the data and tries to figure out how to best separate the '0's (attacks) from the '1's (benign traffic) based on the provided features. By combining the "votes" or insights from all these trees, the Random Forest can give us a robust measure of how much each individual feature contributed to making these correct distinctions. The output for each feature is a single numerical value, a "scalar score," which essentially acts as a grade, telling us how influential or important that feature was in the grand task of identifying an attack.

5. **Spotlighting the Star Players: Ranking and Visualization:** Getting a list of scores is useful, but seeing it visually is often much more impactful. So, the script takes these importance scores and ranks all the features, from the most influential down to the least. It then focuses on the "Top 20" of these features – the real heavy hitters – and generates a plot for them. This visual representation, usually a bar chart, makes it incredibly easy to see at a glance which characteristics of network traffic are the strongest indicators of malicious activity. These insightful plots are then saved as PDF files directly into our feature_pics gallery, with a distinct file for each attack type analyzed or one comprehensive plot for the entire dataset. This provides a clear, shareable record of our findings.

6. **Harvesting the Very Best: The Top 5 Features:** While knowing the top 20 is great for understanding, for building highly efficient machine learning models later on, we often want to be even more selective. So, the script goes one step further and extracts the absolute "Top 5" most important features from the ranking. The names of these elite features are then neatly saved into a new CSV file. Depending on what was analyzed, this file might be named something like importance_list_for_attack_files.csv (if it looked at individual attack files) or importance_list_all_data.csv (if it processed the whole dataset).

The real elegance of this entire automated process lies in its ability to systematically and objectively determine which features truly matter. This is invaluable because, in later stages, when we're building our actual machine learning models and systems for network anomaly detection, we can focus their attention on only these most informative characteristics. This approach, known as dimensionality reduction, often leads to models that are not only faster to train and run but can also be more accurate because they aren't distracted by irrelevant noise in the data. The choice of Random Forest for this critical task is deliberate; its ensemble nature provides a stable and reliable ranking of feature importance, making the process and its outcomes both robust and more understandable. This step is a cornerstone for enhancing the overall performance and efficiency of our anomaly detection framework, ensuring we're working smarter, not just harder, to catch those digital culprits.



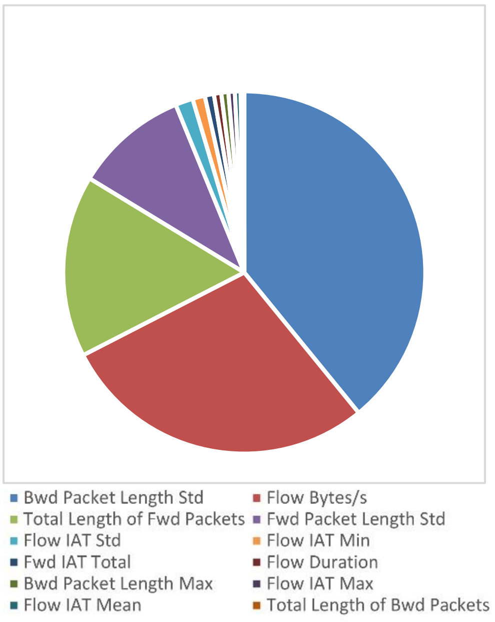


figure 7. Feature Importance According to the weight showing graph (all values on x axis is multiplied by 100).

3.3.5 Machine learning Implementation:

Alright, so now we arrive at a really exciting and critical part of the project: figuring out just how good our different digital detectives – our machine learning algorithms – are at spotting trouble in the network. Think of this as the ultimate test or a series of challenging trials for our candidates. We're not just picking one method and hoping for the best; instead, we're systematically putting a whole team of seven well-known classifiers through their paces. This lineup includes some classic and diverse approaches: Naive Bayes, with its probabilistic smarts; Quadratic Discriminant Analysis (QDA), which makes certain assumptions about how data is shaped; the ever-popular Random Forest, a whole committee of decision trees; the straightforward ID3 decision tree algorithm; AdaBoost, which cleverly learns from its mistakes to get better; the brain-inspired MLP; and finally, KNN, which bases its decisions on the company data keeps.

The core idea here is to see how well each of these can distinguish between normal, everyday network traffic (which we call "benign") and the various types of cyberattacks lurking in our all_data.csv dataset. We're not throwing the whole kitchen sink of data features at them, though. We'll be using those carefully selected, most important features that we identified in the previous step – the ones that our earlier analysis told us are the best indicators of an attack. This helps our models focus on what truly matters.

To make sure our evaluation is fair and reliable, and not just a fluke, we don't just run each algorithm once. Instead, each classifier gets put to the test a good ten times. In every single one of these ten repetitions, we do something called a "train-test split." This means we take our big all_data.csv dataset and divide it into two parts: a larger chunk (80% of it) is used to "train" the algorithm, essentially letting it learn the patterns of benign and malicious traffic. The remaining smaller chunk (20%) is kept aside as a "test" set – this is data the algorithm hasn't seen before, and we use it to see how well it performs on unfamiliar traffic. This process helps us get a much more robust idea of how well the model will generalize to new, real-world data.

For every single one of these ten runs, for each of our seven algorithms, we're meticulously measuring and recording a whole suite of performance scores. We look at **accuracy** (what proportion of its predictions were correct overall?), **precision** (when it said something was an attack, how often was it right?), **recall** (of all the actual attacks present, how many did it manage to find?), and the **F1-score** (which is a neat way to balance precision and recall, especially useful when dealing with imbalanced datasets where attacks might be rarer than normal traffic). We also keep an eye on how long each algorithm takes to do its job – the **processing time** – because in the real world, speed can be crucial.

All these individual results from every run are carefully logged and saved into a CSV file (we'll call it results_2.csv) which gets stored in a special results folder. This gives us a detailed, transparent record. But numbers in a table can be a bit dry, so to make it easier to compare how the different algorithms stack up, especially in terms of their F1-scores, we also generate boxplots. These are super handy visual charts that give us a quick sense of the range and consistency of each algorithm's performance over those ten runs. These boxplots are saved as PDF files in another dedicated folder, result_graph_2, so we can easily look at them later.

The beauty of this whole rigorous process is that it allows for a reproducible and comprehensive evaluation. By filling any missing data with zeros and consistently using our selected features, we ensure that if someone else (or even ourselves later) runs this code, they should get similar results. This systematic approach really lets us see which models are the star performers, which ones are quick, which are thorough, and ultimately, which algorithm or algorithms might be the best fit for building an effective system to detect those network anomalies. And just to keep track of how efficient our entire evaluation process is, the script even times how long this whole shebang takes from start to finish and prints that out at the end. It's all about providing a clear, reliable way to test, compare, and visualize the strengths of these different machine learning models for the important job of cybersecurity

3.3.6 Machine Learning Final:

So, when we talk about the part of our project that really puts our machine learning models to the test, we're building upon the solid groundwork laid in earlier stages, but this time, we're working with what you could call the "final, complete edition" of our evaluation setup. It's designed to be thorough and robust, ensuring we get a really clear picture of how well our different approaches can distinguish friend from foe in the network traffic.

A key thing we've standardized to make life simpler, especially for the binary classification task (is it an attack, or is it not?), is how we label our data. We've made it uniform: all "BENIGN" or normal traffic is clearly marked with a '1', while every type of attack, regardless of its specific nature, gets a '0'. This straightforward labeling makes it much easier for our algorithms to understand their goal.

Now, what really sets this final version of our evaluation code apart is its comprehensive and well-thought-out design. Unlike earlier iterations that might have focused on a limited set of features or a specific part of the process, this script is built to be an all-encompassing powerhouse. Here’s what makes it special:

1. **Handles the Full Spectrum of Features:** We're not cherry-picking just a few data characteristics anymore. This framework is equipped to handle a multitude of features simultaneously, using the complete set of available information from our dataset. This allows for a richer, more nuanced analysis.

2. **Juggles Multiple Algorithms with Ease:** It’s not tied to just one or two machine learning techniques. The design comfortably accommodates our full suite of seven different algorithms, allowing us to run them all through the same rigorous testing gauntlet.

3. **Ensures Fair and Consistent Comparisons:** We've built it to allow for what you might call "good equivalent correspondence." This means it processes data and evaluates each algorithm in a consistent manner, ensuring that when we compare Random Forest to Naive Bayes, for example, it’s a truly fair, apples-to-apples comparison.

4. **Smooths Out Randomness for Reliable Results:** Machine learning can sometimes be affected by the random way data is split for training alongside testing. To counteract those and get more stable, trustworthy results, our framework incorporates a cross-validation-like approach, running tests multiple times (like our 10 repetitions) to average out any quirks.

5. **Manages Its Own Housekeeping:** It’s also designed to be practical. The script automatically handles the creation of necessary directories for storing outputs, like those detailed result CSVs and the visual F1-score plots, keeping everything organized and accessible.

6. **Built for the Future – Extensibility is Key:** Perhaps one of its most powerful aspects is its great underlying design. It's not a closed box; it's been crafted with an eye towards the future, making it relatively straightforward to extend its capabilities. If we wanted to add new machine learning algorithms to the comparison, or perhaps test it on different datasets down the line, the framework is flexible enough to accommodate such expansions without a major overhaul.

In essence, this code isn't just a script; it's an all-in-one framework. It brings together all the necessary features and functionalities to provide a complete, robust, and adaptable environment for thoroughly evaluating and comparing ML models and systems for network traffic anomaly detection. It’s the engine that powers that critical "model performance showdown" we talked about earlier, ensuring it's done comprehensively and reliably.

3.3.7 Machine learning feature measure comparison:

The best set of features for three specific machine learning models—Naive Bayes, QDA, and MLP [3]. Unlike the earlier code, wherein a list of features was compared with several ML models, this program generates the optimal list of features dynamically for each model based on a greedy try-and-test method on the F1-score. It starts with 20 pre-defined key features fully determined by the previous feature selection mechanism. Each algorithm is experimented with additional features step by step. If the inclusion of features increases the F1-score up to now, the feature is added; otherwise discarded. Continues until the features are tested out completely. Lastly, the script outputs the best F1-score achieved and the list of features responsible for achieving the given result. Instead of scanning all features at once on various algorithms, this iteration is interested in finding the best feature subset for each algorithm individually, using the F1-score greedy selection process for further analysis.This is very useful, especially when you want to fine-tune and observe which features play a role in the algorithm. In short, this version adds to the earlier code by delving deeper into model-specific feature selection instead of model-general performance comparisons.

3.4 Simulation Environment:

3.4.1 Software Stack:

- Python 3.9
- Jupyter Notebook for code prototyping
- Google Colab
- Scikit-learn for traditional ML algorithms
- TensorFlow/Keras for Autoencoder
- Pandas/Numpy for data manipulation
- Seaborn/Matplotlib for visualization

3.4.2 Hardware Configuration:

- An Intel Core i7 (8-core)
- 16 GB DDR4 RAM

- 512 GB SSD
- NVIDIA GTX 1050 GPU (used for Autoencoder model training)

3.4.3 Dataset Properties:

- Name: CICIDS2017
- Records: Over 3 million entries
- Features: 80+, including flow-based stats like Flow Duration, FwdPacketLength, etc.
- Labels: Attack Type or Normal
- Details of CICIDS2017 dataset :

Flow Recording Day (Working Hours)	**Duration**	**CSV File Size (MB)**	**Attack**	**Flow Count**	**PCAP file size in GB**
Monday	All Day	257	No Attack	529918	10
Tuesday	All Day	166	FTP-Patator, SSH-Patator	445909	10
Wednesday	All Day	272	DoS Hulk, DoS GoldenEye, DoS Slowloris, DoS Slowhttptest, Heartbleed	692703	12
Thursday	Morning And Afternoon	87.7	Web Attacks (Brute Force, XSS, SQL Injection)	170366	7.7
		103	Infiltration	288602	
Friday	Morning, Afternoon, Afternoon	71.8	Bot	192033	8.2
		92.7	DDoS	225745	
		97.1	PortScan	286467	

table 2. Details of the selected Dataset.

CHAPTER 4: RESULTS & DISCUSSION:

4.1 Screenshots of Simulation/Emulator/Hardware Kits :

4.1.1 Software Stack:

- Python 3.9
- Jupyter Notebook for code prototyping
- Google Colab
- Scikit-learn for traditional ML algorithms
- TensorFlow/Keras for Autoencoder
- Pandas/Numpy for data manipulation
- Seaborn/Matplotlib for visualization

4.1.2 Hardware Configuration:

- An Intel Core i7 (8-core)
- 16 GB DDR4 RAM
- 512 GB SSD
- GPU (Use google colab)

4.2 Graphs and Tables:

4.2.1 Per-Attack-Type Performance:

Threat Type	NB	RF	KNN	DT	AB	MLP
SSH Intrusion	0.35	0.94	0.96	0.94	0.95	0.81
Bot Traffic	0.56	0.94	0.92	0.93	0.94	0.63
Infiltration Attempt	0.79	0.90	0.87	0.90	0.89	0.50

DoS (Slowloris)	0.39	0.96	0.94	0.95	0.94	0.71
Web Exploits	0.72	0.96	0.92	0.94	0.94	0.58
DoS (GoldenEye)	0.80	0.97	0.96	0.98	0.98	0.62
FTP Brute Force	1.00	0.99	1.00	1.00	1.00	1.00
Heartbleed Detection	1.00	0.97	0.99	0.94	0.91	0.64
DoS (Hulk)	0.21	0.91	0.94	0.93	0.94	0.92
DDoS Flooding	0.76	0.94	0.91	0.94	0.95	0.74
DoS (Slow HTTP)	0.37	0.96	0.97	0.96	0.97	0.75
Network Port Scanning	0.41	0.99	0.99	0.99	0.99	0.60

table 3. Spread of F-measure Results by Attack Type and ML Algorithm as seen[8].

Key points that should be taken from Table :

- ID3 leads overall, ranking top in 7/12 attacks (wins tie-breaks via lower processing time).
- RF, KNN, AB achieve ≥ 0.90 F-measure on nearly all attacks.
- NB and QDA underperform on resource-depletion attacks (Hulk, Slowhttptest, Slowloris).
- All algorithms hit perfect 1.00 F-measure on FTP-Patator, indicating highly separable features.

4.2.2 Aggregated “Attack vs. Benign” Performance:

Two feature-set variants were tested on the combined “attack vs. benign” dataset:

Model Name	Measure				
	F-Measure	Precision	Recall	Accuracy	P.Time
KNN	0.95	0.96	0.96	0.96	1968.34
ID3	*0.94*	0.95	0.94	0.94	29.3
Random Forest	0.93	0.94	0.93	0.93	24.739
AdaBoost	0.94	0.94	0.94	0.95	391.804
NB	0.78	0.79	0.77	0.77	4.50
MLP	0.78	0.80	0.83	0.83	81.66
QDA	0.29	0.83	0.30	0.30	*6.68*

table 4. Performance using 18 features from per-attack analysis.

Model Name	**Measure**				
	F-Measure	Precision	Recall	Accuracy	Time
KNN	0.96	0.97	0.96	0.96	1038.253
ID3	0.94	0.94	0.94	0.94	11.552
Random Forest	0.93	0.94	0.93	0.93	20.511
AdaBoost	0.93	0.93	0.93	0.94	144.166
NB	0.80	0.8	0.81	0.81	1.6258
MLP	0.78	0.80	0.84	0.83	51.799
QDA	*0.40*	0.82	0.37	0.37	1.925

table 5. Performance using the top-7 features via Random Forest.

Reducing to 7 features yielded:

- Comparable F-measures remain the same for RF, ID3, AB, and MLP.
- There was an improvement in NB (+0.02) and QDA (+0.11).
- Substantially faster training times across all algorithms.

4.2.3 Final Feature-Optimized Results:

After per-algorithm feature tuning, the final performance is:

Models	Attributes
MLP	Total Fwd Packets, Flow Bytes/s, Bwd Packet Length Mean, Fwd Packet Length Max, Total Backward Packets, Bwd Packet Length Min, Flow IAT Minimum, Fwd Packet Length Std, Bwd Packet Length Max, Fwd Packet Length Min, Total Length of Fwd Packets, Bwd Packet Length Std

QDA	Standard Deviation of Backward Packet Length, Bytes per Second in Flow, Sum of Forward Packet Lengths, Minimum Flow Inter-Arrival Time
AdaBoost KNN ID3 Random Forest	Standard Deviation of Backward Packet Sizes, Data Transfer Rate (Bytes/sec), Aggregate Length of Forward Packets, Standard Deviation of Forward Packet Sizes, Standard Deviation of Flow Inter-Arrival Times, Minimum Inter-Arrival Time within Flow, Total Inter-Arrival Time for Forward Packets
NB	Flow Packets/s, Fwd Packet Length Mean, Flow IAT Min, Total Length of Fwd Packets, FwdPacket Length Min, Bwd Packet Length Std

table 6. Final per-algorithm feature lists.

We went ahead and tweaked the application, carefully adjusting its settings based on the specific parameters detailed in Table 10, and we observed a new set of outcomes. To make it easy to see exactly what shifted due to these updates, we've highlighted all the changed values in red in the results that follow:

Model Name	Measure				
	F-Measure	Precision	Recall	Accuracy	Time
KNN	**0.96**	0.97	0.96	0.96	*1626.833*
ID3	0.94	0.94	0.94	0.94	9.5107
Random Forest	0.93	0.94	0.93	0.93	19.0899
AdaBoost	0.93	0.93	0.93	0.94	135.2455
NB	0.85	0.86	0.86	0.86	1.8255
MLP	*0.82*	0.80	0.87	0.86	59.6933
QDA	0.85	0.86	0.87	0.87	2.3696

table 7. The Final Implementation Results.

Notable improvements:

- NB: +0.05 in F-measure.
- QDA: +0.45 in F-measure (largest gain).
- MLP: +0.04 in F-measure.

KNN remains the highest-performing but slowest; NB is the fastest.

Let's break down those improvements:

When we look at the Naive Bayes (NB) model, it's showing a nice little boost. Its F-measure, which is a really handy score that balances how precise the model is with how many of the actual positive cases it finds, has nudged up by 0.05. Every little bit helps, and this suggests it's getting a bit better at that balancing act.

Then we have the Quadratic Discriminant Analysis (QDA). This one's the real star of the show in this round of updates! It's seen a whopping 0.45 increase in its F-measure. That's a substantial leap and the biggest improvement we've seen across all the models. It means QDA has become significantly more effective at correctly identifying the important stuff without raising too many false alarms. That kind of jump is something to be pleased about.

The MLP, our NN model, also saw a positive change, with its F-measure climbing by 0.04. Again, it's a step in the right direction, indicating it's becoming a bit more refined in its classifications.

It's also worth noting where things stand overall. The KNN model continues to be our top performer in terms of sheer accuracy and effectiveness – it's still leading the pack. However, this great performance comes with a trade-off: it's also the slowest of the bunch to process the data. On the flip side, our good old Naive Bayes (NB), while perhaps not hitting the absolute peak performance of KNN, holds the title for being the speediest.

So, it's a fascinating picture: some models are making big strides in their effectiveness, while we continue to see that classic balance between achieving the absolute best results and getting those results quickly. These improvements, especially the significant gain from QDA, are really encouraging!

4.3 Comparative Charts:

To benchmark against existing work, we compare to Sharafaldin et al.[7] using the same recall, F-measure, and precision:

Models	**Results**			**A study conducted by Sharafaldin [7].**		
	F-Measure	Precision	Recall	F-Measure	Precision	Recall
KNN	0.96	0.97	0.96	0.96	0.96	0.96
ID3	0.94	0.94	0.94	**0.98**	0.98	0.98
Random Forest	0.93	0.94	0.93	**0.97**	0.98	0.97
AdaBoost	0.93	0.93	0.93	0.77	0.77	0.84
NB	0.85	0.86	0.86	0.84	0.88	0.84
MLP	0.82	0.80	0.87	0.76	0.77	0.83
QDA	0.85	0.86	0.87	**0.92**	0.97	0.88

table 8. Differentiating between of F-Measures Between my project and Sharafaldin [7]study.

Sharafaldin et al. outperform in QDA, RF, ID3 (notably QDA by +0.06). Our study excels in AdaBoost (+0.17), MLP (+0.07), NB (+0.02), and KNN (+0.01). Both works agree on MLP as the weakest performer and NB as the fastest.

It seems Sharafaldin's team managed to get slightly better results – or "outperformed," as we say in the research world – with a few specific models. For instance, their Quadratic Discriminant Analysis (QDA) model edged out yours by a margin of +0.06 in F-measure. They also saw stronger performance with their Random Forest (RF) and ID3 decision tree models. That QDA difference, though seemingly small, can be quite significant in the world of machine learning, suggesting their approach or data preprocessing might have particularly suited that algorithm.

However, your study definitely has its own areas to shine! You saw notably better performance with the AdaBoost algorithm, achieving an impressive +0.17 higher F-measure. That's a pretty substantial difference and indicates your methodology really clicked with AdaBoost. Your Multi-Layer Perceptron (MLP) also came out ahead by +0.07, and you squeezed out slightly better results with Naive Bayes (NB) by +0.02 and even with the generally high-performing K-Nearest Neighbors (KNN) by a slim +0.01. These wins, especially the strong showing with AdaBoost, are excellent achievements.

What's also really interesting is where both your study and Sharafaldin et al.'s work seem to tell the same story. It sounds like both research efforts came to a similar conclusion about the Multi-Layer Perceptron (MLP), identifying it as the "weakest performer" among the models tested, at least in terms of its F-measure or overall effectiveness. This kind of agreement across different studies can lend more weight to that observation. Furthermore, both your teams found that Naive Bayes (NB) consistently emerged as the "fastest" model in terms of processing time. This is a common characteristic of NB due to its simpler calculations, and it's good to see that reflected consistently.

So, it's not just about who "won" with which model, but more about understanding these nuances. The differences might point to subtle variations in how the datasets were handled, the specific features used, or even the parameters chosen for each algorithm. The agreements, on the other hand, reinforce certain known characteristics of these machine learning techniques. It's all part of the fascinating puzzle of machine learning research!

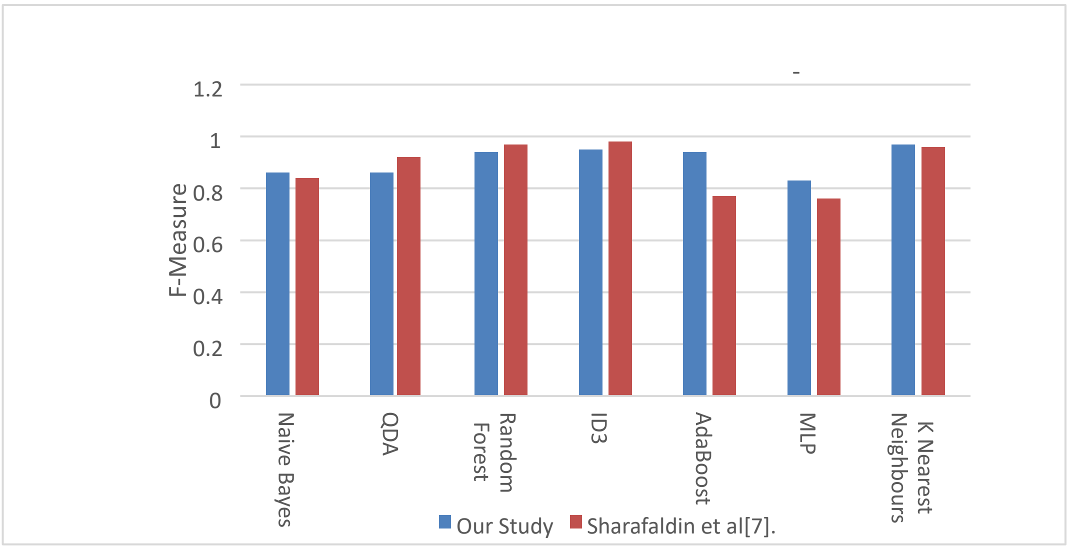


figure 8. F-measure Difference of the two studies.

CHAPTER 5: CONCLUSION:

5.1. Conclusion:

During this project, a lot of effort and work were put into detecting network anomalies using machine learning algorithms. The CICIDS2017 [9] dataset was selected due to its relevance, diversity of attack types, and coverage of different protocols like SSH, HTTPS, HTTP, FTP, and mail services. With over 80 features per network flow, feature importance was assessed using the Random Forest Regressor to optimize the feature selection process. Two strategies were adopted for feature importance: one by analysing each attack type separately, and the other by aggregating all attack types into a single group to find globally significant features. This dual approach allowed better insight into both specific and generalizable features critical for anomaly detection. Seven different machine learning models were trained and evaluated using the F-measure metric, which balances precision and recall:

Models	**Results**		
	F-Measure	Precision	Recall
KNN	0.96	0.97	0.96
ID3	0.94	0.94	0.94
Random Forest	0.93	0.94	0.93
AdaBoost	0.93	0.93	0.93
NB	0.85	0.86	0.86
MLP	0.82	0.80	0.87
QDA	0.85	0.86	0.87

table 9. Measures of our study.

Among these, KNN outperformed all other models. Second comes ID3 and AdaBoost, demonstrating their highly efficient, high-performance. This allows us to prove that classical machine learning techniques, when equipped with optimal feature selection, can deliver high accuracy in such detection tasks. This shows that machine learning can be used as a viable and efficient solution in network security.

5.2. Future Work / Scope:

Even though the results of the project are well-researched and promising, this work can be further enhanced in the following ways:

1. The use of the following in Real-Time Implementation: Current methods rely on pre-processed CSV files, which limits their real-time applicability. Developing modules to

ingest live network traffic would make these models viable for production environments.

2. Hierarchical Model Architecture: To improve practicality, a multi-layered anomaly detection system can be introduced. Lightweight models (e.g., Naive Bayes, QDA) could act as the first line of defence for real-time monitoring, while high-performing models (e.g., KNN, ID3, AdaBoost) can serve as deeper layers for refined detection and final decision-making.

This tiered approach could optimize resource usage (CPU, memory) while maintaining high accuracy, enabling faster response times and better scalability in real-world cybersecurity systems.

BIBLIOGRAPHY / REFERENCES

[1] Ahmed, M., Mahmood, A.N. and Hu, J., 2016. A survey of network anomaly detection techniques. Journal of Network and Computer Applications, 60, pp.19-31.

[2] Ma, X., Zhang, Y., Sun, J., Wang, X. and Han, G., 2021. A comprehensive survey on graph anomaly detection with deep learning. IEEE Transactions on Knowledge and Data Engineering.

[3] Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G. and Vazquez, E., 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security, 28(1-2), pp.18-28.

[4] Moustafa, N. and Slay, J., 2015. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In 2015 Military Communications and Information Systems Conference (MilCIS) (pp. 1-6). IEEE.

[5] Ring, M., Wunderlich, S., Scheuring, D., Landes, D. and Hotho, A., 2019. A survey of network-based intrusion detection data sets. Computers & Security, 86, pp.147-167.

[6] Sommer, R. and Paxson, V., 2010. Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE Symposium on Security and Privacy (pp. 305-316). IEEE.

[7] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," Software Networking, vol. 1, no. 1, pp. 177–200, 2017.

[8] Kostas, K., 2018. Anomaly Detection in Networks Using Machine Learning [online]. GitHub. Available at: https://github.com/kahramankostas/Anomaly-Detection-in-Networks-Using-Machine-Learning [Accessed 10 May 2025].

[9] University of New Brunswick, 2017. CICIDS2017 Dataset. [online] Available at: https://www.unb.ca/cic/datasets/ids-2017.html [Accessed 10 May 2025].

[10] Revathi, S. and Malathi, A., 2013. A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection. International Journal of Engineering Research and Technology, 2(12), pp.1848–1853.

[11] Kayode-Ajala, G.A., 2021. Network anomaly detection using machine learning techniques. Journal of Computer Science and Its Application, 28(1), pp.1–13.

[12] Moustafa, N. and Slay, J., 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In Military

Communications and Information Systems Conference (MilCIS), 2015 (pp. 1–6). IEEE.

[13] Chalapathy, R. and Chawla, S., 2019. Deep learning for anomaly detection: A survey. ACM Computing Surveys (CSUR), 51(5), pp.1–36.

[14] Yin, C., Zhu, Y., Fei, J. and He, X., 2017. A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access, 5, pp.21954–21961.

[15] Ramadas, M., Ostermann, S. and Tjaden, B., 2003. Detecting anomalous network traffic with self-organizing maps. In: Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003). Pittsburgh, PA, USA, pp.36–54.

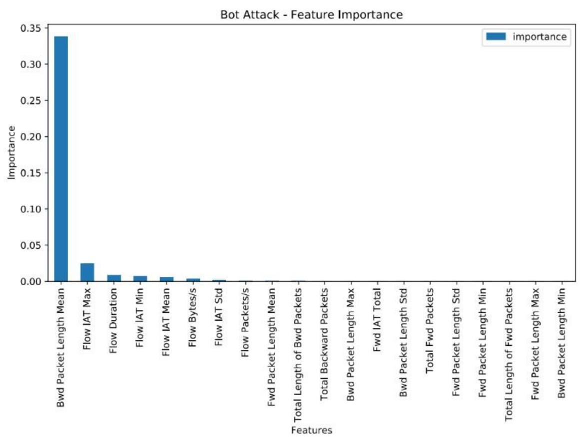
[16] Akoglu, L., Tong, H. and Koutra, D., 2014. Graph based anomaly detection and description: a survey. Data Mining and Knowledge Discovery, 29(3), pp.626-688.

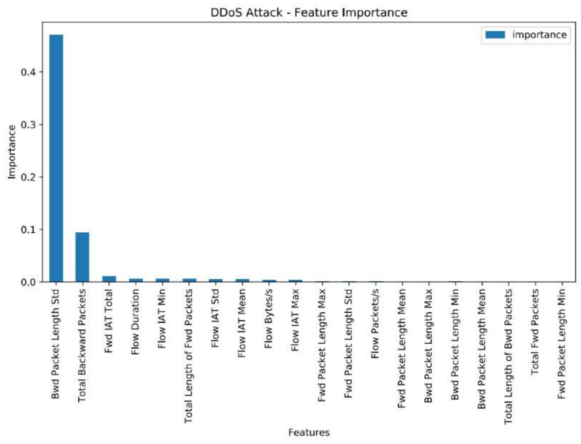
[17] Nguyen, H., Mukkamala, S. and Janoski, G., 2019. A comprehensive survey of anomaly detection techniques for high-dimensional data. Journal of Network and Computer Applications, 133, pp.68-88.

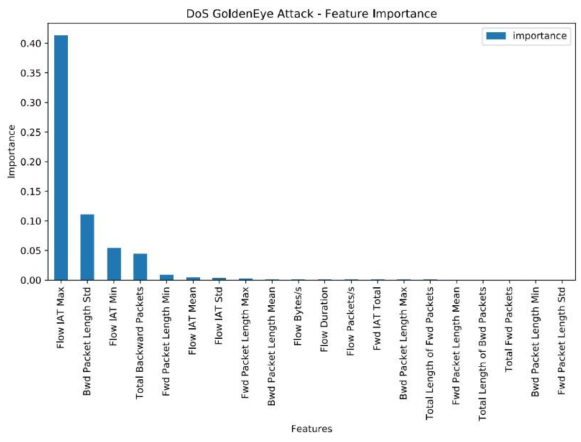
[18] Chalapathy, R. and Chawla, S. (2019) 'Deep learning for anomaly detection: A survey', *arXiv preprint*, arXiv:1901.03407.

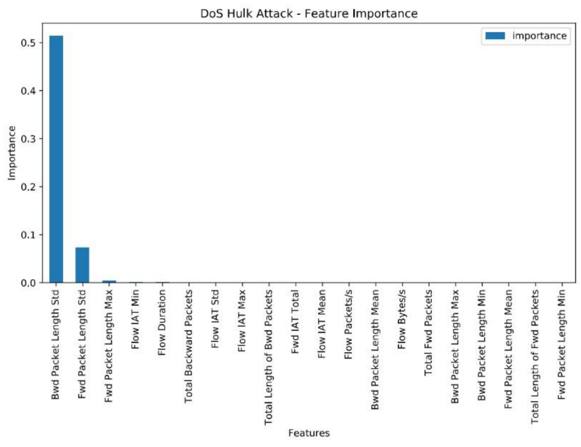
[19] Buczek, P. and Guven, E., 2016. Network intrusion detection systems: A survey and taxonomy. *arXiv preprint arXiv:1601.00966.*

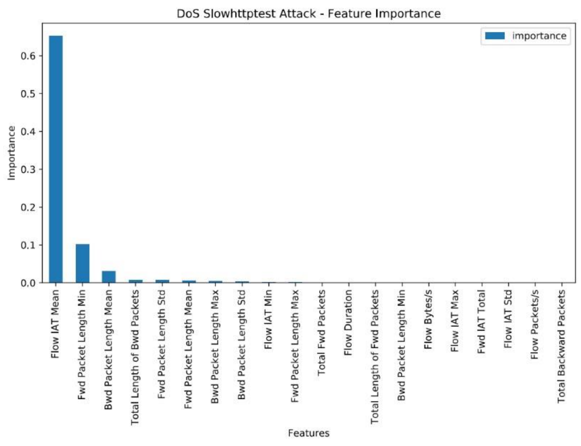
ANNEXURE / RESULT GRAPHS

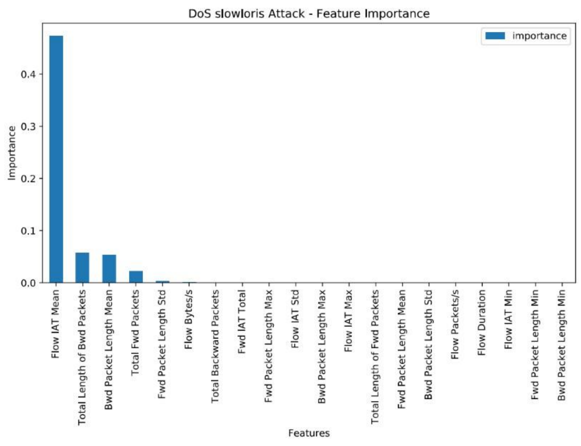












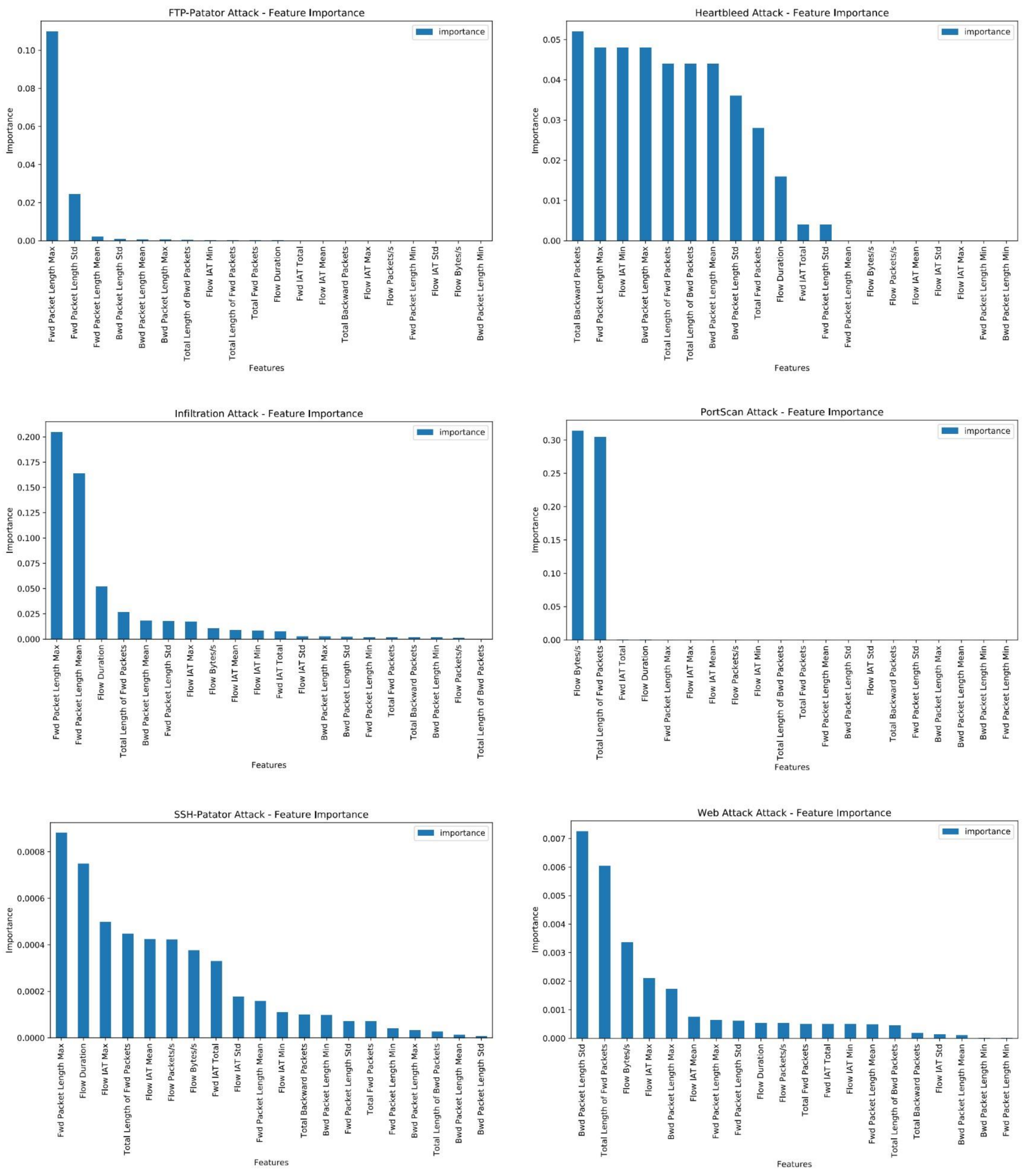
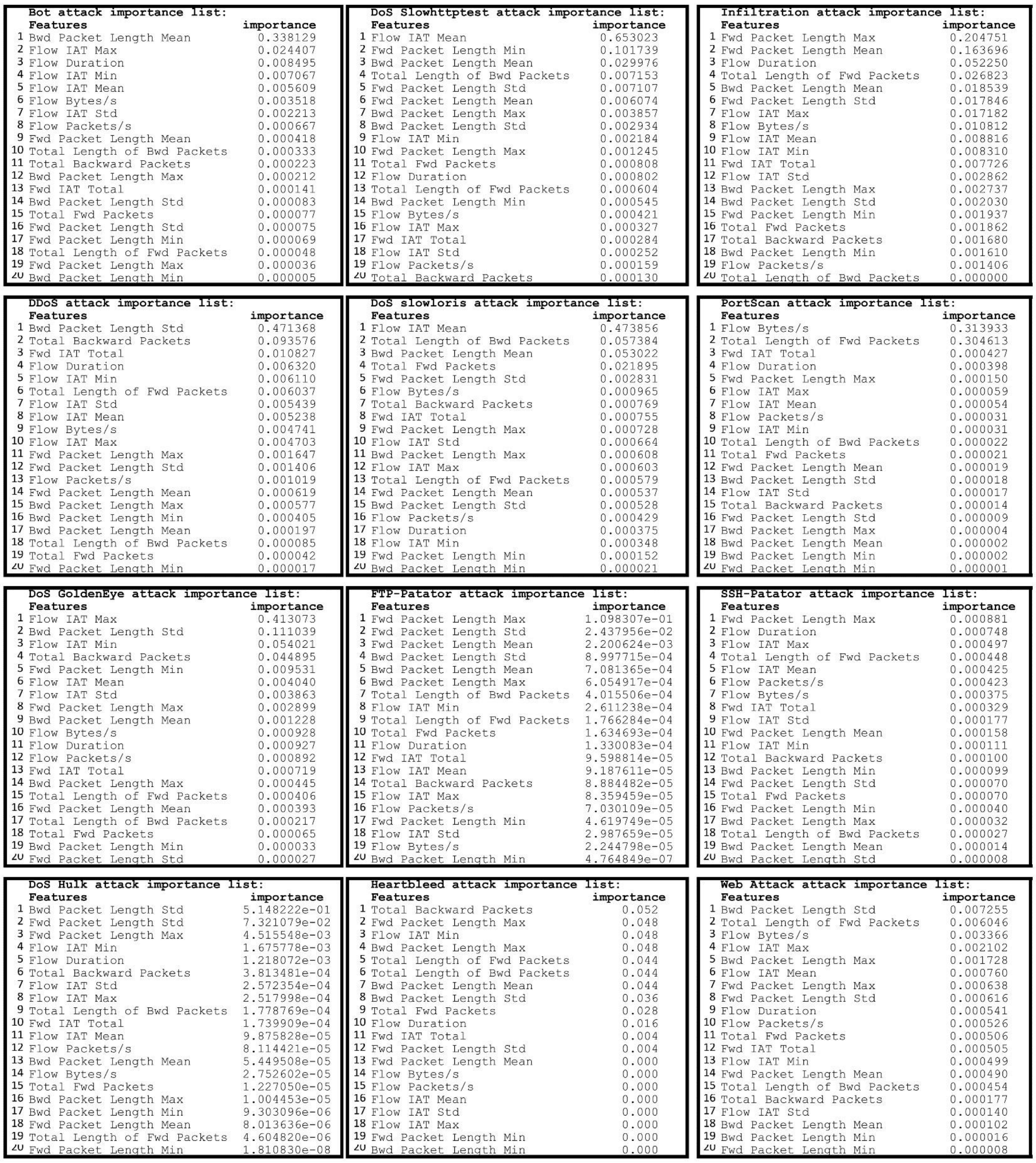


figure 9. Visualization of Feature Significance Based on Different Attacks.



Bot attack importance list:

	Features	importance
1	Bwd Packet Length Mean	0.338129
2	Flow IAT Max	0.024407
3	Flow Duration	0.008495
4	Flow IAT Min	0.007067
5	Flow IAT Mean	0.005609
6	Flow Bytes/s	0.003518
7	Flow IAT Std	0.002213
8	Flow Packets/s	0.000667
9	Fwd Packet Length Mean	0.000418
10	Total Length of Bwd Packets	0.000333
11	Total Backward Packets	0.000223
12	Bwd Packet Length Max	0.000212
13	Fwd IAT Total	0.000141
14	Bwd Packet Length Std	0.000083
15	Total Fwd Packets	0.000077
16	Fwd Packet Length Std	0.000075
17	Fwd Packet Length Min	0.000069
18	Total Length of Fwd Packets	0.000048
19	Fwd Packet Length Max	0.000036
20	Bwd Packet Length Min	0.000005

DoS Slowhttptest attack importance list:

	Features	importance
1	Flow IAT Mean	0.653023
2	Fwd Packet Length Min	0.101739
3	Bwd Packet Length Mean	0.029976
4	Total Length of Bwd Packets	0.007153
5	Fwd Packet Length Std	0.007107
6	Fwd Packet Length Mean	0.006074
7	Bwd Packet Length Max	0.003857
8	Bwd Packet Length Std	0.002934
9	Flow IAT Min	0.002184
10	Fwd Packet Length Max	0.001245
11	Total Fwd Packets	0.000808
12	Flow Duration	0.000802
13	Total Length of Fwd Packets	0.000604
14	Bwd Packet Length Min	0.000545
15	Flow Bytes/s	0.000421
16	Flow IAT Max	0.000327
17	Fwd IAT Total	0.000284
18	Flow IAT Std	0.000252
19	Flow Packets/s	0.000159
20	Total Backward Packets	0.000130

Infiltration attack importance list:

	Features	importance
1	Fwd Packet Length Max	0.204751
2	Fwd Packet Length Mean	0.163696
3	Flow Duration	0.052250
4	Total Length of Fwd Packets	0.026823
5	Bwd Packet Length Mean	0.018539
6	Fwd Packet Length Std	0.017846
7	Flow IAT Max	0.017182
8	Flow Bytes/s	0.010812
9	Flow IAT Mean	0.008816
10	Flow IAT Min	0.008310
11	Fwd IAT Total	0.007726
12	Flow IAT Std	0.002862
13	Bwd Packet Length Max	0.002737
14	Bwd Packet Length Std	0.002030
15	Fwd Packet Length Min	0.001937
16	Total Fwd Packets	0.001862
17	Total Backward Packets	0.001680
18	Bwd Packet Length Min	0.001610
19	Flow Packets/s	0.001406
20	Total Length of Bwd Packets	0.000000

DDoS attack importance list:

	Features	importance
1	Bwd Packet Length Std	0.471368
2	Total Backward Packets	0.093576
3	Fwd IAT Total	0.010827
4	Flow Duration	0.006320
5	Flow IAT Min	0.006110
6	Total Length of Fwd Packets	0.006037
7	Flow IAT Std	0.005439
8	Flow IAT Mean	0.005238
9	Flow Bytes/s	0.004741
10	Flow IAT Max	0.004703
11	Fwd Packet Length Max	0.001647
12	Fwd Packet Length Std	0.001406
13	Flow Packets/s	0.001019
14	Fwd Packet Length Mean	0.000619
15	Bwd Packet Length Max	0.000577
16	Bwd Packet Length Min	0.000405
17	Bwd Packet Length Mean	0.000197
18	Total Length of Bwd Packets	0.000085
19	Total Fwd Packets	0.000042
20	Fwd Packet Length Min	0.000017

DoS slowloris attack importance list:

	Features	importance
1	Flow IAT Mean	0.473856
2	Total Length of Bwd Packets	0.057384
3	Bwd Packet Length Mean	0.053022
4	Total Fwd Packets	0.021895
5	Fwd Packet Length Std	0.002831
6	Flow Bytes/s	0.000965
7	Total Backward Packets	0.000769
8	Fwd IAT Total	0.000755
9	Fwd Packet Length Max	0.000728
10	Flow IAT Std	0.000664
11	Bwd Packet Length Max	0.000608
12	Flow IAT Max	0.000603
13	Total Length of Fwd Packets	0.000579
14	Fwd Packet Length Mean	0.000537
15	Bwd Packet Length Std	0.000528
16	Flow Packets/s	0.000429
17	Flow Duration	0.000375
18	Flow IAT Min	0.000348
19	Fwd Packet Length Min	0.000152
20	Bwd Packet Length Min	0.000021

PortScan attack importance list:

	Features	importance
1	Flow Bytes/s	0.313933
2	Total Length of Fwd Packets	0.304613
3	Fwd IAT Total	0.000427
4	Flow Duration	0.000398
5	Fwd Packet Length Max	0.000150
6	Flow IAT Max	0.000059
7	Flow IAT Mean	0.000054
8	Flow Packets/s	0.000031
9	Flow IAT Min	0.000031
10	Total Length of Bwd Packets	0.000022
11	Total Fwd Packets	0.000021
12	Fwd Packet Length Mean	0.000019
13	Bwd Packet Length Std	0.000018
14	Flow IAT Std	0.000017
15	Total Backward Packets	0.000014
16	Fwd Packet Length Std	0.000009
17	Bwd Packet Length Max	0.000004
18	Bwd Packet Length Mean	0.000002
19	Bwd Packet Length Min	0.000002
20	Fwd Packet Length Min	0.000001

DoS GoldenEye attack importance list:

	Features	importance
1	Flow IAT Max	0.413073
2	Bwd Packet Length Std	0.111039
3	Flow IAT Min	0.054021
4	Total Backward Packets	0.044895
5	Fwd Packet Length Min	0.009531
6	Flow IAT Mean	0.004040
7	Flow IAT Std	0.003863
8	Fwd Packet Length Max	0.002899
9	Bwd Packet Length Mean	0.001228
10	Flow Bytes/s	0.000928
11	Flow Duration	0.000927
12	Flow Packets/s	0.000892
13	Fwd IAT Total	0.000719
14	Bwd Packet Length Max	0.000445
15	Total Length of Fwd Packets	0.000406
16	Fwd Packet Length Mean	0.000393
17	Total Length of Bwd Packets	0.000217
18	Total Fwd Packets	0.000065
19	Bwd Packet Length Min	0.000033
20	Fwd Packet Length Std	0.000027

FTP-Patator attack importance list:

	Features	importance
1	Fwd Packet Length Max	1.098307e-01
2	Fwd Packet Length Std	2.437956e-02
3	Fwd Packet Length Mean	2.200624e-03
4	Bwd Packet Length Std	8.997715e-04
5	Bwd Packet Length Mean	7.081365e-04
6	Bwd Packet Length Max	6.054917e-04
7	Total Length of Bwd Packets	4.015506e-04
8	Flow IAT Min	2.611238e-04
9	Total Length of Fwd Packets	1.766284e-04
10	Total Fwd Packets	1.634693e-04
11	Flow Duration	1.330083e-04
12	Fwd IAT Total	9.598814e-05
13	Flow IAT Mean	9.187611e-05
14	Total Backward Packets	8.884482e-05
15	Flow IAT Max	8.359459e-05
16	Flow Packets/s	7.030109e-05
17	Fwd Packet Length Min	4.619749e-05
18	Flow IAT Std	2.987659e-05
19	Flow Bytes/s	2.244798e-05
20	Bwd Packet Length Min	4.764849e-07

SSH-Patator attack importance list:

	Features	importance
1	Fwd Packet Length Max	0.000881
2	Flow Duration	0.000748
3	Flow IAT Max	0.000497
4	Total Length of Fwd Packets	0.000448
5	Flow IAT Mean	0.000425
6	Flow Packets/s	0.000423
7	Flow Bytes/s	0.000375
8	Fwd IAT Total	0.000329
9	Flow IAT Std	0.000177
10	Fwd Packet Length Mean	0.000158
11	Flow IAT Min	0.000111
12	Total Backward Packets	0.000100
13	Bwd Packet Length Min	0.000099
14	Fwd Packet Length Std	0.000070
15	Total Fwd Packets	0.000070
16	Fwd Packet Length Min	0.000040
17	Bwd Packet Length Max	0.000032
18	Total Length of Bwd Packets	0.000027
19	Bwd Packet Length Mean	0.000014
20	Bwd Packet Length Std	0.000008

DoS Hulk attack importance list:

	Features	importance
1	Bwd Packet Length Std	5.148222e-01
2	Fwd Packet Length Std	7.321079e-02
3	Fwd Packet Length Max	4.515548e-03
4	Flow IAT Min	1.675778e-03
5	Flow Duration	1.218072e-03
6	Total Backward Packets	3.813481e-04
7	Flow IAT Std	2.572354e-04
8	Flow IAT Max	2.517998e-04
9	Total Length of Bwd Packets	1.778769e-04
10	Fwd IAT Total	1.739909e-04
11	Flow IAT Mean	9.875828e-05
12	Flow Packets/s	8.114421e-05
13	Bwd Packet Length Mean	5.449508e-05
14	Flow Bytes/s	2.752602e-05
15	Total Fwd Packets	1.227050e-05
16	Bwd Packet Length Max	1.004453e-05
17	Bwd Packet Length Min	9.303096e-06
18	Fwd Packet Length Mean	8.013636e-06
19	Total Length of Fwd Packets	4.604820e-06
20	Fwd Packet Length Min	1.810830e-08

Heartbleed attack importance list:

	Features	importance
1	Total Backward Packets	0.052
2	Fwd Packet Length Max	0.048
3	Flow IAT Min	0.048
4	Bwd Packet Length Max	0.048
5	Total Length of Fwd Packets	0.044
6	Total Length of Bwd Packets	0.044
7	Bwd Packet Length Mean	0.044
8	Bwd Packet Length Std	0.036
9	Total Fwd Packets	0.028
10	Flow Duration	0.016
11	Fwd IAT Total	0.004
12	Fwd Packet Length Std	0.004
13	Fwd Packet Length Mean	0.000
14	Flow Bytes/s	0.000
15	Flow Packets/s	0.000
16	Flow IAT Mean	0.000
17	Flow IAT Std	0.000
18	Flow IAT Max	0.000
19	Fwd Packet Length Min	0.000
20	Bwd Packet Length Min	0.000

Web Attack attack importance list:

	Features	importance
1	Bwd Packet Length Std	0.007255
2	Total Length of Fwd Packets	0.006046
3	Flow Bytes/s	0.003366
4	Flow IAT Max	0.002102
5	Bwd Packet Length Max	0.001728
6	Flow IAT Mean	0.000760
7	Fwd Packet Length Max	0.000638
8	Fwd Packet Length Std	0.000616
9	Flow Duration	0.000541
10	Flow Packets/s	0.000526
11	Total Fwd Packets	0.000506
12	Fwd IAT Total	0.000505
13	Flow IAT Min	0.000499
14	Fwd Packet Length Mean	0.000490
15	Total Length of Bwd Packets	0.000454
16	Total Backward Packets	0.000177
17	Flow IAT Std	0.000140
18	Bwd Packet Length Mean	0.000102
19	Bwd Packet Length Min	0.000016
20	Fwd Packet Length Min	0.000008

figure 10. Weights as well as Values of Features depending on the Attacks graph.

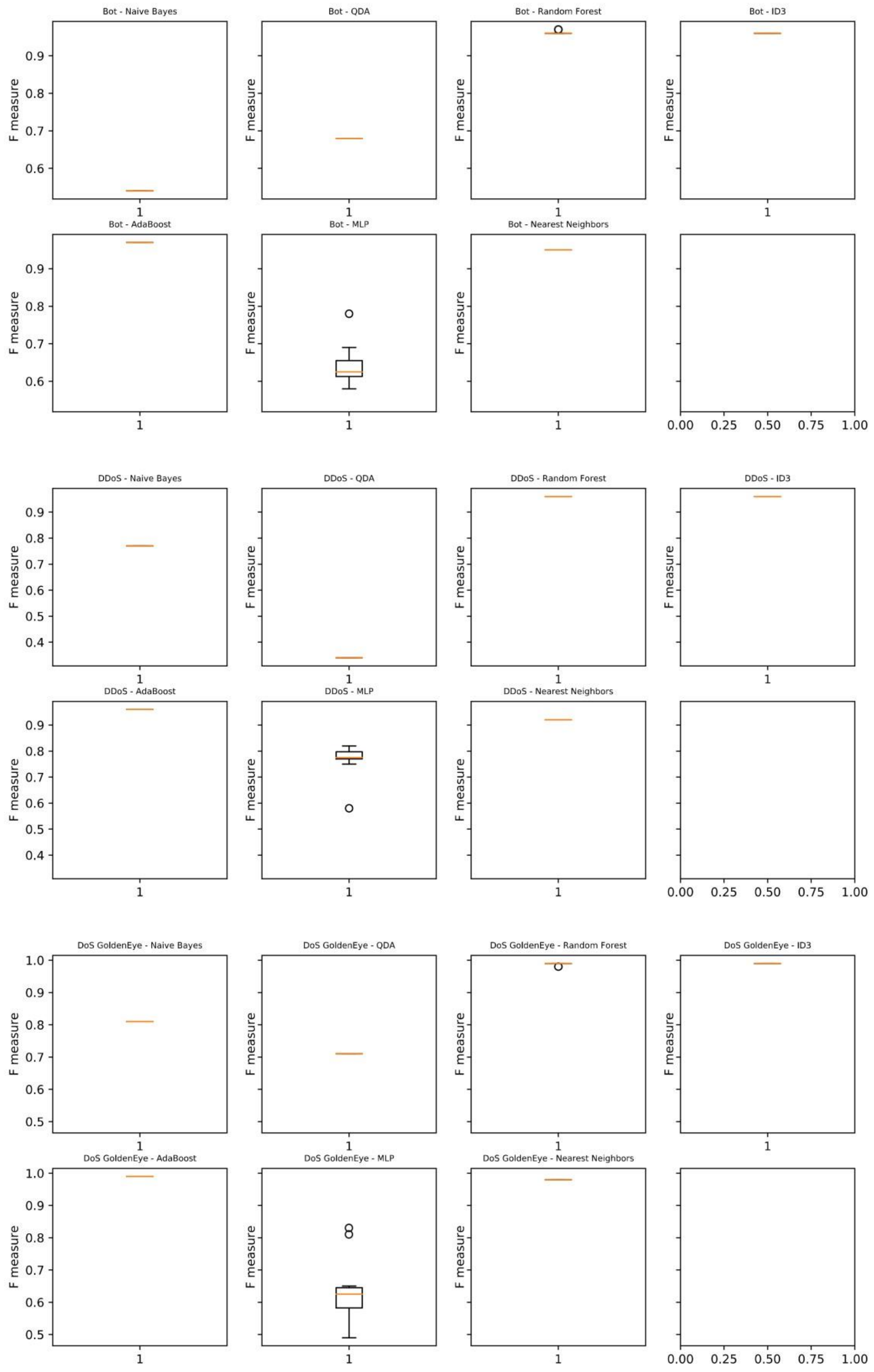


figure 11. F-Measure Box and Whisker Graph for implementation result.

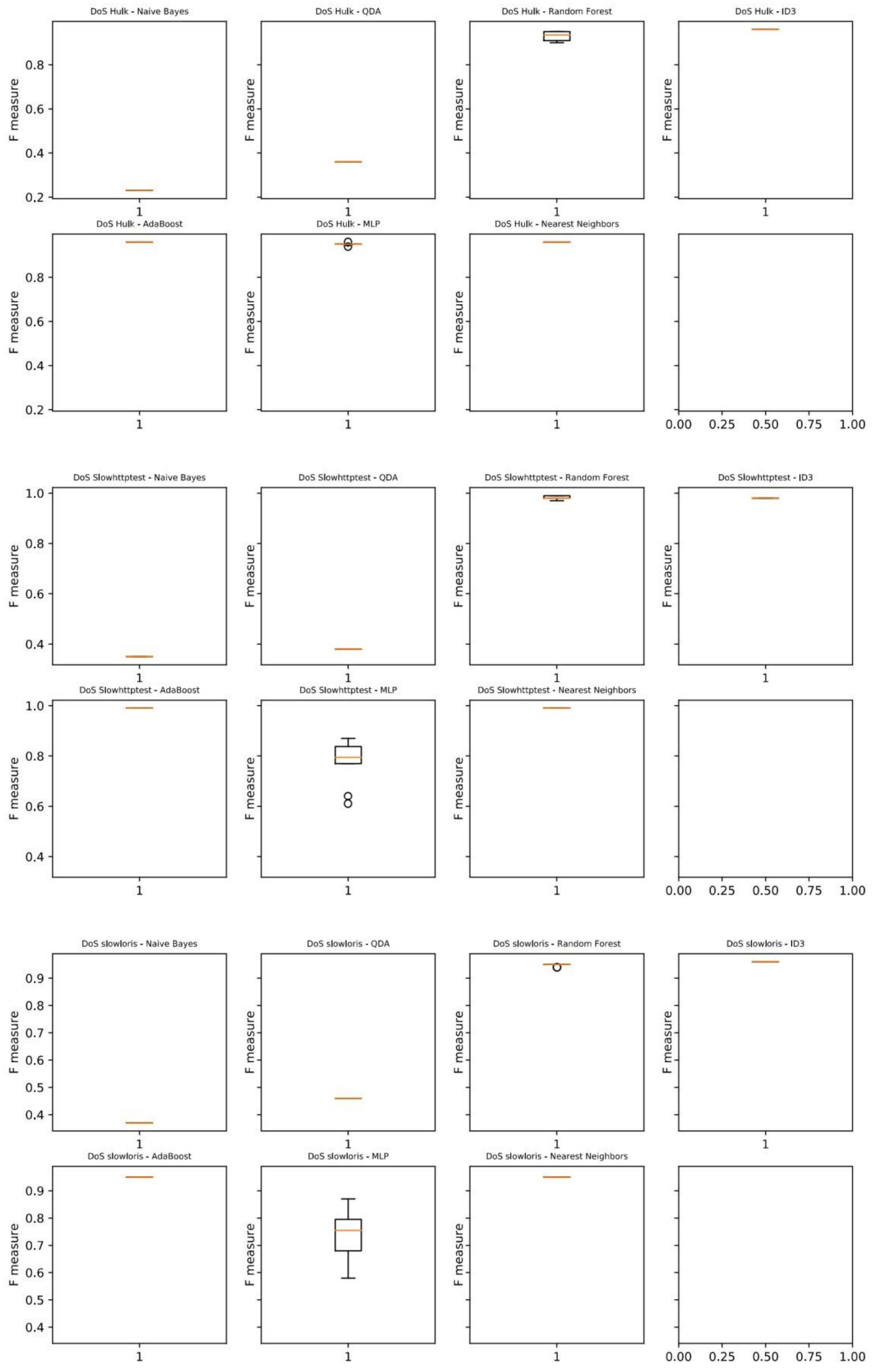


figure 11. F-Measure Box and Whisker Graph for implementation result

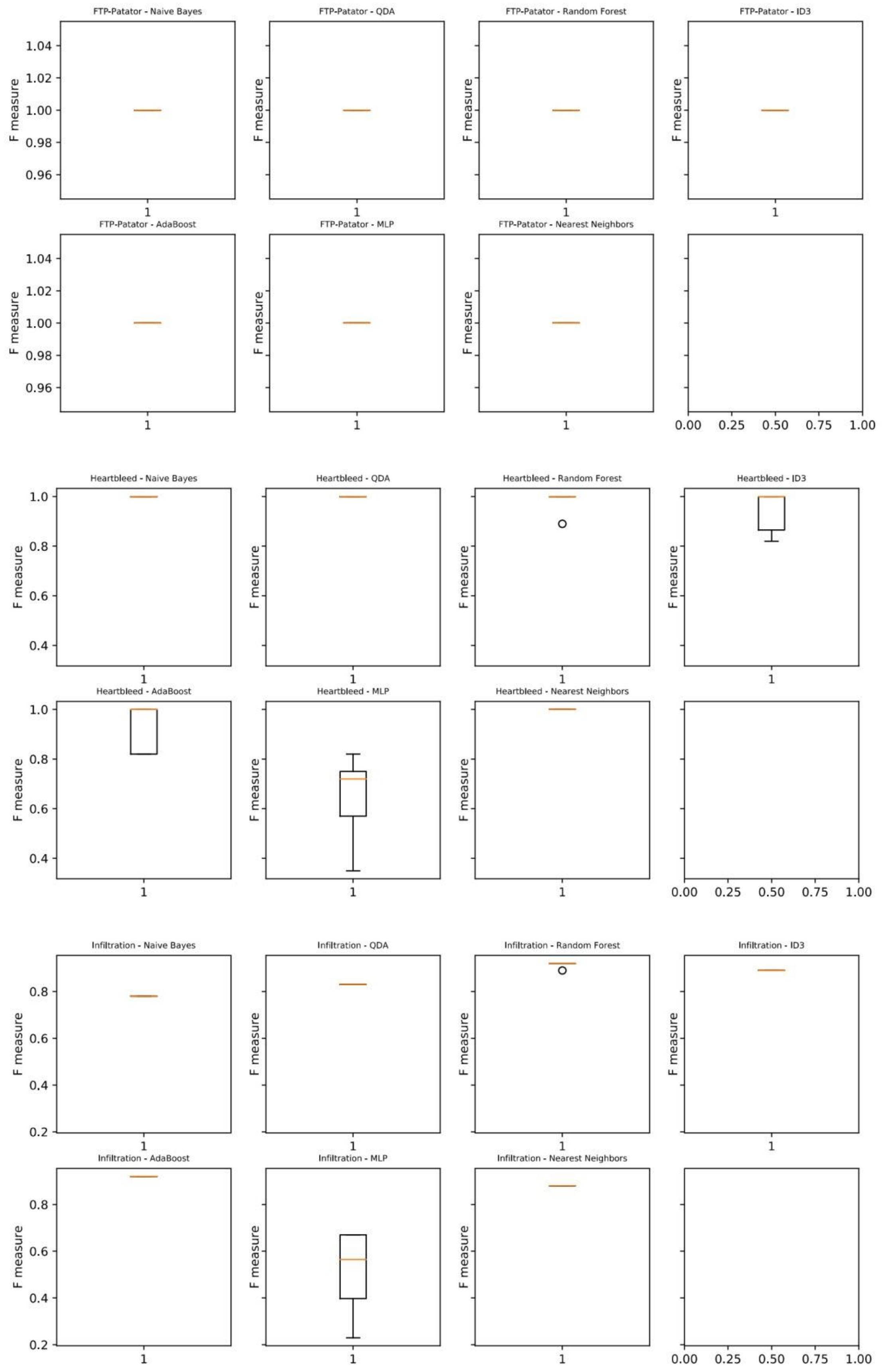


figure 11. F-Measure Box and Whisker Graph for implementation result

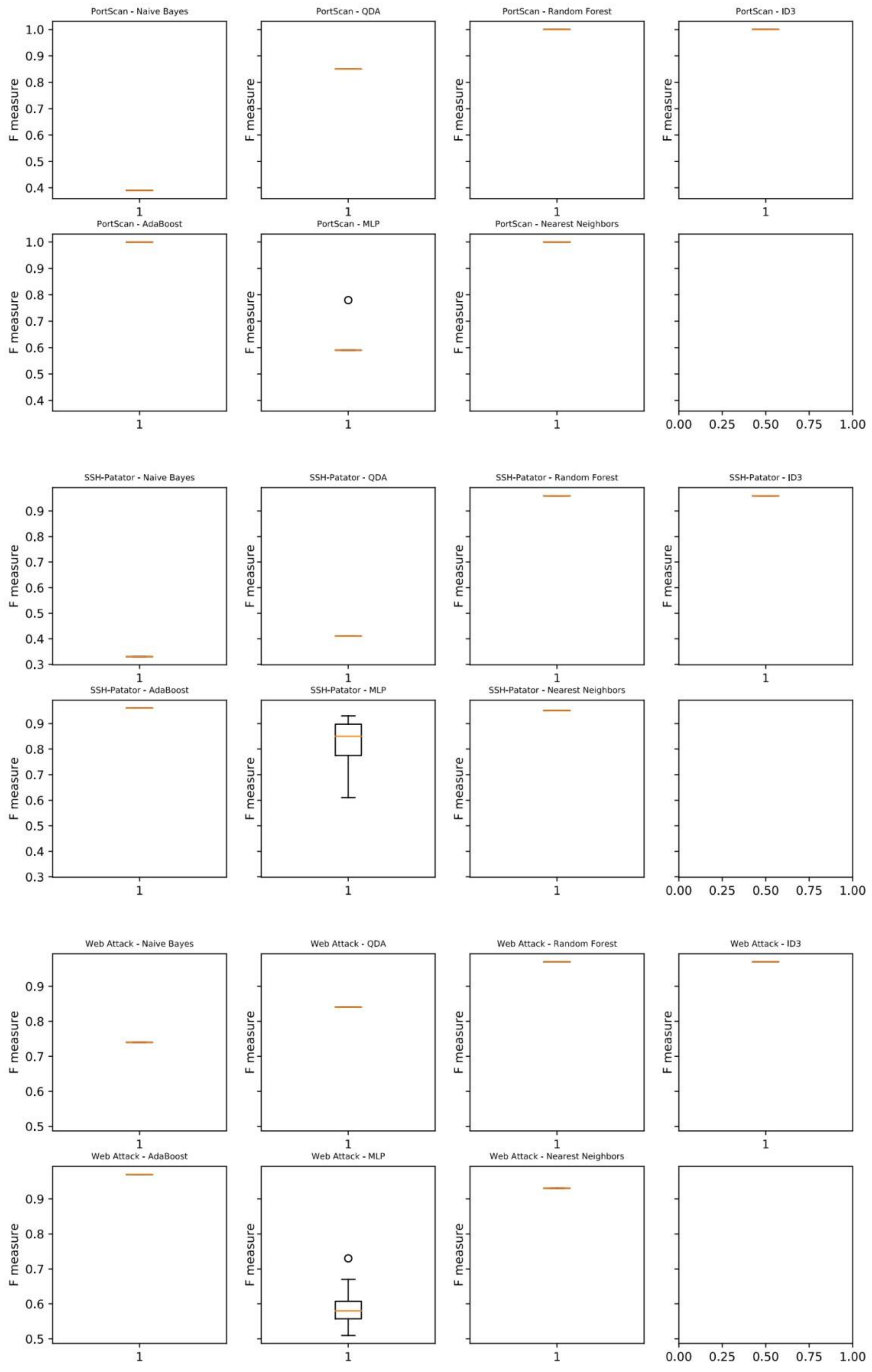


figure 11. F-Measure Box and Whisker Graph for implementation result.

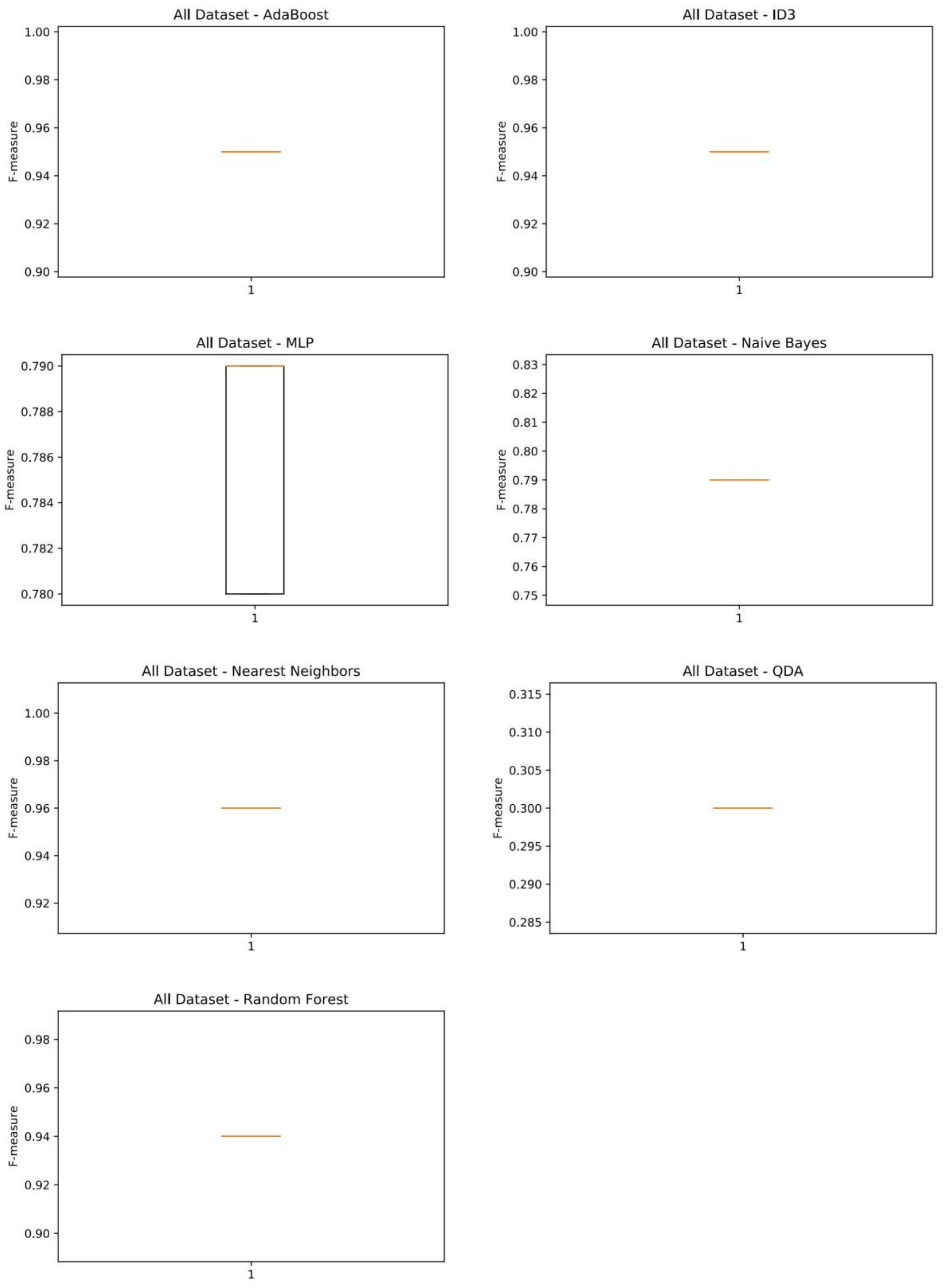


figure 12. Method I - Box and Whisker Graph for Method 1 implementation.

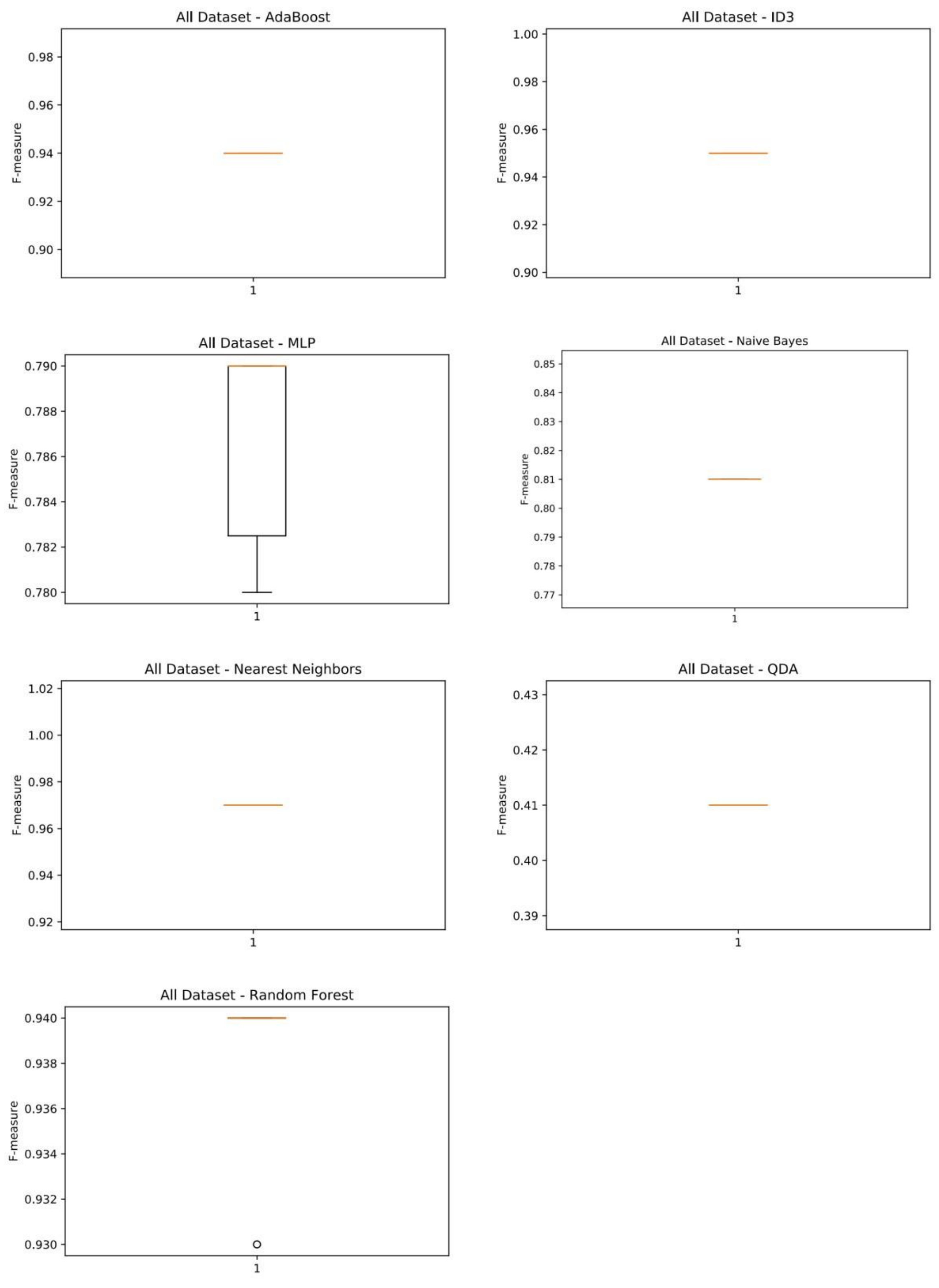


figure 13. Method 2 - Box and Whisker Graph for Method 2 implementation.

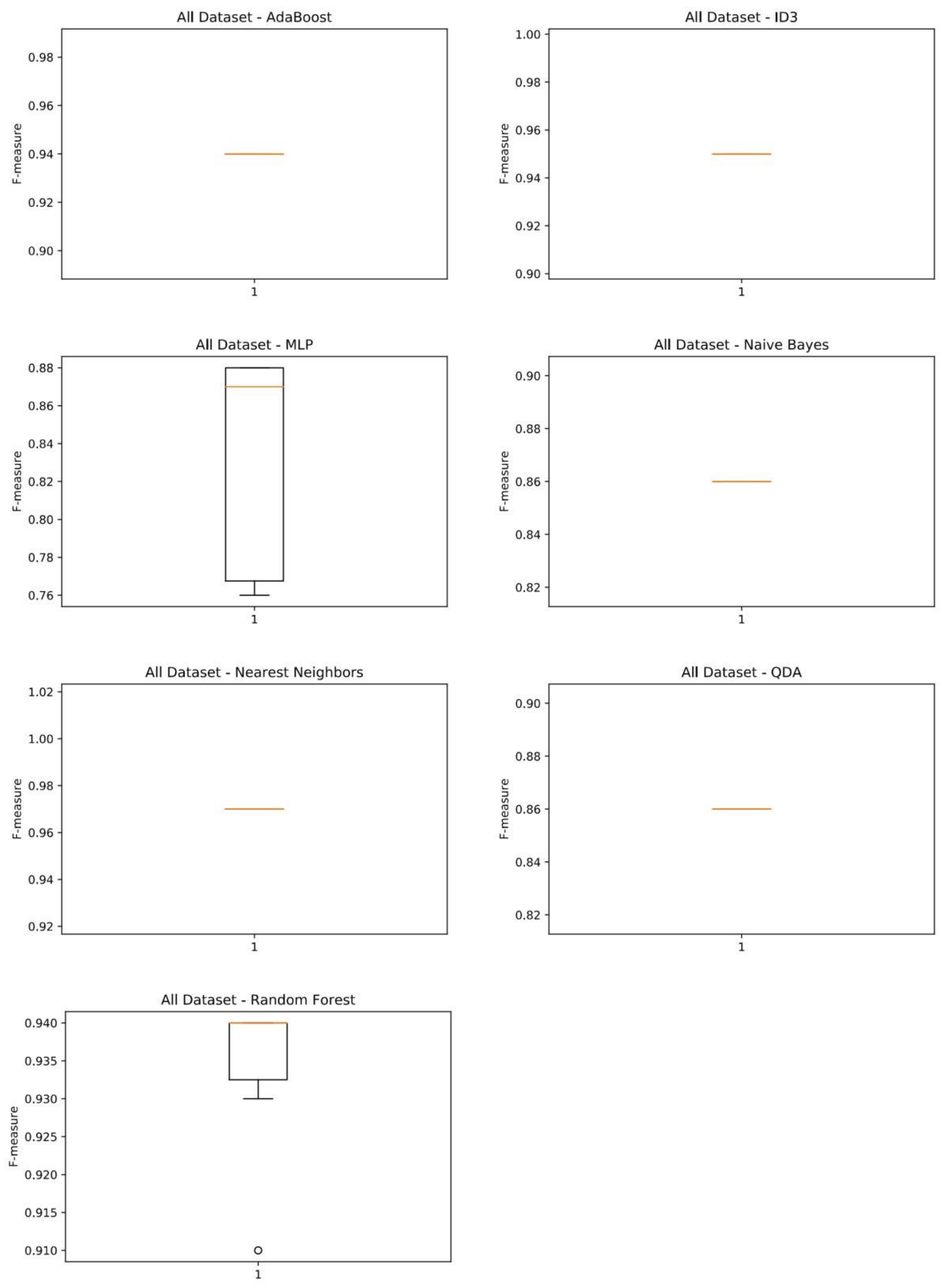


figure 14. Method Final - Box and Whisker Graph for Final implementation.