TÜRK-ALMAN ÜNİVERSİTESİ
TÜRKISCH-DEUTSCHE UNIVERSITÄT

T.C.
TURKISH-GERMAN UNIVERSITY
INSTITUTE OF THE GRADUATE STUDIES
IN SCIENCE AND ENGINEERING

REAL-TIME MACHINE LEARNING ANOMALY DETECTION IN COMPUTER NETWORKS

Master's Thesis

Halit Canap DEMİR

ISTANBUL 2024

TÜRK-ALMAN ÜNİVERSİTESİ
TÜRKISCH-DEUTSCHE UNIVERSITÄT

T.C.

TURKISH-GERMAN UNIVERSITY

INSTITUTE OF THE GRADUATE STUDIES

IN SCIENCE AND ENGINEERING

Master's Thesis

Halit Canap DEMİR

M.Sc., Computer Science and Engineering, Turkish-German University, 2024

Advisor
Asst. Prof. Dr. Ziya Cihan TAYŞİ

Submitted to the Institute of the Graduate Studies in
Science and Engineering in partial fulfillment of the requirements for the
Master's Degree

ISTANBUL 2024

REAL-TIME MACHINE LEARNING ANOMALY DETECTION IN COMPUTER NETWORKS

APPROVED BY:

Assistant Prof. Dr. Z. Cihan TAYŞİ
(Thesis Advisor)

Prof. Dr. A. Gökhan YAVUZ
Turkish-German University

Assistant Prof. Dr. Ö. Özgür BOZKURT
Fenerbahçe University

DATE OF APPROVAL: 15 November 2024

ACKNOWLEDGMENTS

This journey of growth and discovery has been truly transformative, and I have been fortunate to receive immense support from numerous individuals and institutions, to whom I express my deepest gratitude.

I would like to express my heartfelt thanks to my thesis advisor, Asst. Prof. Dr. Ziya Cihan TAYŞİ, whose guidance, patience, and encouraging approach have been invaluable throughout this journey. His mentorship not only advanced my research skills but also contributed significantly to my personal growth. I am truly grateful for his constant support, which helped me navigate through the challenges of this work with greater confidence and clarity. I am also sincerely thankful to Prof. Dr. Ali Gökhan YAVUZ for introducing me to Dr. TAYŞİ, whose mentorship has significantly shaped the direction and quality of this work.

My heartfelt thanks go to my parents, Hülya and Hayrettin, whose unwavering encouragement and belief in my abilities have been my greatest motivation. To my siblings—Onur, Zafer, Emre, Ömer, Yusuf, and my twin sister Nursima—thank you for your endless love and support. Growing up with you has been a constant source of strength and joy, and I am thankful for each one of you.

I am also deeply grateful to my close friend and colleague, Kerim ATMACA, for his unending support and encouragement throughout this process. His friendship and insight have been a crucial part of this journey, and I am truly appreciative of his presence, both in times of challenge and success.

Lastly, I want to thank Prof. Dr. Ali Gökhan YAVUZ and Asst. Prof. Dr. Özgür BOZKURT for their presence and valuable suggestions during my thesis defense as committee members.

To everyone mentioned above, and to those who have impacted my journey in ways both big and small, I am deeply grateful. Your support and influence have made this path not only possible but profoundly meaningful.

DECLARATION OF AUTHENTICITY

I declare that I completed the master thesis independently and used only the materials that are listed. All materials used, from published as well as unpublished sources, whether directly quoted or paraphrased, are duly reported. Furthermore, I declare that the master's thesis, or any abridgment of it, was not used for any other degree-seeking purpose and give the publication rights of the thesis to the Institute of the Graduate Studies in Science and Engineering, Turkish-German University.

Signature
Halit Canap DEMİR
05.11.2024

ABSTRACT

REAL-TIME MACHINE LEARNING ANOMALY DETECTION IN COMPUTER NETWORKS

In the rapidly evolving digital world, the need for advanced security measures to protect our data has steadily increased. The growing cyber threats have made it essential to develop sophisticated Intrusion Detection Systems (IDS) that can adapt to modern network environments. In this thesis, to address this need, a system that detects malicious traffic by analyzing network traffic flows using deep learning methods is proposed. Various datasets that could be used for system development were examined, and the CICIDS2017 dataset, which stands out in terms of relevance and scope, was chosen. The CICIDS2017 dataset contains a total of 15 classes, one representing normal network traffic and the others representing different types of attacks. Training the deep learning model with a consistent and balanced dataset directly impacts system performance. Therefore, pre-processing steps such as removing missing or redundant data, eliminating irrelevant features, and balancing the number of examples in different classes were performed. Dense Neural Networks (DNN) and Random Forest (RF), methods commonly used in similar studies, were selected for the proposed model. The models developed could detect network traffic involving different types of attacks with an average accuracy of 98.5%. The main goal of this study is to detect attacks on the network. Accordingly, a version of the dataset consisting of two classes—normal network traffic and attack traffic—was created. Using this dataset, another system was developed that could detect malicious traffic with 98.8% accuracy. The systems developed in this thesis aim to detect attacks in real-time within a network. Therefore, after optimizing performance through experiments with different parameters, the models were tested in a real network environment using the NVIDIA Jetson AGX Orin embedded system. For the sustainability of the developed system, training with current network traffic and attacks is also essential. In this regard, the training of the models on the embedded system was analyzed in terms of time and performance.

Keywords: *Intrusion Detection Systems (IDS), Real-time IDS, CICIDS2017 Dataset, Machine Learning Models, Deep Learning Models, Multiclass and Binary Classification, Dense Neural Networks (DNN), Random Forest Classifiers, NVIDIA Jetson AGX Orin*

ÖZET

BİLGİSAYAR AĞLARINDA GERÇEK ZAMANLI MAKİNE ÖĞRENİMİ ANOMALİ TESPİTİ

Hızla gelişen dijital dünyada, verilerimizi korumak için gelişmiş güvenlik önlemlerine duyulan ihtiyacı giderek artırdı. Artan siber tehditler, modern ağ ortamlarına uyum sağlayabilen sofistike Saldırı Tespit Sistemlerinin (Intrusion Detection System, IDS) geliştirilmesini zorunlu hale getirmiştir. Tez kapsamında bu ihtiyacı karşılamak amacıyla ağ içerisindeki trafik akışlarının derin öğrenme yöntemleri ile incelenerek zararlı trafiğin tespit edildiği bir sistem önerilmiştir. Sistemin geliştirilmesi için kullanılabilecek çeşitli veri setleri incelenmiş ve güncellik ve kasam acısından öne çıkan CICIDS2017 veri setinin kullanılması tercih edilmiştir. CICIDS2017 veri seti içerisinde biri normal ağ trafiğine diğeri ise farklı tipteki saldırılara ait olmak üzere toplam 15 adet sınıf bulunmaktadır. Oluşturulacak derin öğrenme modelinin tutarlı ve dengeli bir veri seti ile eğitilmesi Sistem başarımı üzerinde doğrudan etkilidir. Bu nedenle veri seti üzerinde eksik veya tekrarlı verinin silinmesi, önemsiz özelliklerin çıkarılması, farklı sınıflardaki örnek sayısının dengelenmesi gibi ön işlemler gerçekleştirilmiştir. Oluşturulacak model için benzer çalışmalarda yaygın şekilde kullanılan Yoğun Sinir Ağları (Dense Neural Network, DNN) ve Rastgele Orman (Random Forest, RF) yöntemlerinin kullanılması tercih edilmiştir. Oluşturulan modeller ile farklı saldırı tiplerine ait ağ trafiği ortalama olarak %98.5 başarım ile tespit edilebilmektedir. Çalışma kapsamında temel hedef ağ üzerindeki saldırıların tespit edilmesidir. Buna bağlı olarak veri setinin normal ağ trafiği ve saldırı trafiği olmak üzere iki sınıftan oluşan bir versiyonu oluşturulmuştur. Bu veri seti üzerinde yapılan çalışmalar ile zararlı trafiğin %98.8 ile tespit edilebildiği bir sistem daha ortaya konmuştur. Tez kapsamında geliştirilen sistemlerin bir ağ içerisindeki saldırıları gerçek zamanlı olarak tespit edilebilmesi hedeflenmektedir. Bu sebeple farklı parametreleri üzerinde yapılan denemeler sonrasında performansı optimize edilen modeller, NVIDIA Jetson AGX Orin gömülü sistemi üzerinde ve gerçek ağ ortamında test edilmiştir. Geliştirilen sistemin devamlılığı açısından güncel ağ trafiği ve saldırılar ile eğitilmesi de önemlidir. Bu kapsamda geliştirilen modellerin gömülü sistem üzerindeki eğitimleri de zaman ve performans açısından incelenmiştir.

Anahtar Sözcükler: *Saldırı Tespit Sistemleri (IDS), Gerçek zamanlı IDS, CICIDS2017 veri seti, Makine öğrenimi modelleri, Derin Öğrenme Modelleri, Çok sınıflı ve ikili sınıflandırma, Yoğun Sinir Ağları (DNN), Rastgele Orman (Random Forest) sınıflandırıcıları, NVIDIA Jetson AGX Orin,*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

EPH	Example Placeholder
ACK	Acknowledge Flag
AGX	Autonomous Machines GPU Accelerated
AI	Artificial Intelligence
A100, L4, T4	NVIDIA GPU models used for model training and testing
B/S	Batch Size
CICIDS2017	Canadian Institute for Cybersecurity Intrusion Detection System 2017 dataset
CSV	Comma-Separated Values
CVE	Common Vulnerabilities and Exposures
CWR	Congestion Window Reduced
DDoS	Distributed Denial of Service
DNN	Dense Neural Network
Dst, Dest	Destination
ECE	Explicit Congestion Notification Echo
ES	Early Stopping
FI	Feature Importance
GPU	Graphics Processing Unit
IAT	Inter-Arrival Time
IPC	Inter-Process Communication
IP	Internet Protocol
LR	Learning Rate
MAWI	Measurement and Analysis on the WIDE Internet
ML	Machine Learning
MLCVE	Machine Learning Dataset Clean Validated Experiment
MLCVE_clean	Multi-class CVE cleaned dataset
MLCVE_clean_dest	Multi-class CVE cleaned dataset without destination port information
MLCVE_binary	MLCVE_clean CVE binary dataset
MLCVE_binary_dest	MLCVE_clean_dest binary dataset without destination port information

PCAP	Packet Capture
ReLU	Rectified Linear Unit
RF	Random Forest
SDD	Software Design Description
SRC	Source
SRS	Software Requirements Specification
SSH	Secure Shell
TCP	Transmission Control Protocol
UNSW-NB15	University of New South Wales Network-Based 2015 dataset

1. INTRODUCTION

1.1. Background

Advancement of technology has always affected the way humans interact with their environment. Different milestones within history contributed differently to these interactions. However, modern technology has completely transformed the practices in which we communicate, especially with the increase in internet access. The increase naturally added hundreds of millions annually into the numbers of internet users[1]. As a result, the Internet traffic volume increased 20.5% annually in the last 5 years[2]. Internet traffic is the flow of data across the entire Internet or specific network connections of its fundamental networks. A network is basically a web of interconnected servers or devices that communicate with each other to share resources. A known and widespread example of a network is the Internet we use. In this context, computer networks constitute the main element in transporting the internet traffic. This transportation creates network traffic and it can be described as the amount of data moving across a computer network at any given time. Given the nature of the network traffic, the data it carries can contain personal, financial and/or corporation information. The confidential information that is carried can be targeted for several reasons by several individuals and groups. This results in what is known as the computer network attacks. It is important to understand what constitutes the computer networks traffic in order to get a gist of how the attacks occur. The basic construction to create a traffic.

The data, which constantly flows between various network nodes, is fragmented into smaller data bits known as network packets or data packets. This fragmentation is essential to allow effective usage of the network's interconnection medium by all computers within the network. Data in a network packet is divided into two components: the packet header and the payload, each with distinct purposes. The packet header is responsible for carrying essential information such as content, host address, and destination address. In contrast, the payload consists of the real data that is being transmitted.

Network packets are distributed across a network through communication protocols that enable the transmission and exchange of data within the extensive, interconnected network of nodes. The Internet Protocol (IP) is a communication protocol that governs the movement of data packets between different nodes in a network using a defined set of rules. TCP (Transmission Control Protocol) is a protocol that ensures reliable and orderly delivery of data over the internet.

Organizations typically employ the Transmission Control Protocol (TCP) in conjunction with IP to guarantee the accurate transmission and receipt of data packets to their intended host destinations. Certain communication protocols may include packet footers alongside packet headers in data packets, serving to provide supplementary information regarding the packet.

When data packets are grouped to create a single sequence as a data stream based on source and destination IP addresses, port numbers and protocol type, it forms the basis for network flow. The Flow is important because it helps identify unusual patterns or traffic volumes which can point out possible security threats or network problems.

Any deviation from normal network behavior that may indicate security threats, faults, or performance issues is an anomaly in computer networks. Analyzing the flow of traffic is the key to detecting the anomalies within the computer networks. The flow-based anomaly detection can be achieved in different ways. We can divide it into three main categories as Rule-Based Detection, Signature-Based Detection and AI-Based Anomaly Detection.

Rule-Based Anomaly Detection is a method that identifies abnormal network flow by using predefined rules and patterns. It compares the observed data with these rules and patterns to detect anomalies for potential threats and problems. The most typical example can be given as Firewalls. It decides whether traffic is allowed based on from which source to which destination with what port it is allowed to communicate. All the communication outside of the defined rules and patterns are blocked. The models that consist of this structure can be divided into two approaches. First one blocks everything and only allows the flow structures that we want. This approach is the most popular one as it is more secure. However the operational workload is higher for this approach because it requires updates for each new change. The second approach is allowing all the traffic and blocking the flows that we

allowed. This approach however results in the system being open to vulnerabilities all the time. Essentially, the primary advantage of rule-based anomaly detection is its precision in identifying specific, well-understood threats. However, rule-based anomaly detection also has limitations, particularly in its ability to identify new or evolving threats. This results in Rule-Based Detection to not be popular for dynamic environments.

Signature-Based Anomaly Detection is an approach that uses identifying the threats by analyzing network traffic against a set of known threat signatures. Signature here means a predefined and known pattern that can help to identify threats based on their unique characteristics. System alerts or takes predefined actions to eliminate the threat when it detects an activity that matches with a threat signature. This is why Signature-Based systems are highly effective against known threat signatures. This aspect also points out the weakness of these systems as they are effective against known threats. They struggle when against new,unknown and evolving threats that do not have similar threat signatures within the system. Signature-Based systems should check and control all the protocols, services and practices in order to keep being up-to-date. Every change should be included into the set of signatures with every change in protocols and applications. This set also keeps maintaining old signatures as they can still be used. After some point this becomes inconvenient and unmanageable. To address shortcomings, these systems can be used with other detection techniques in order to work against unknown threats and zero-day exploits.

At this point, AI-Based Anomaly Detection becomes a leading approach. AI-Based Anomaly Detection uses artificial intelligence techniques like machine learning and deep learning, to identify unusual patterns or behaviors in data that deviate from the normal, often detecting both known and unknown threats more effectively. Unlike traditional rule-based or signature-based methods, AI-based approaches can learn and adapt to the patterns in data. It allows them to detect anomalies that may not have been previously identified. This makes it especially effective against zero-day exploits and complex attacks. Utilizing AI for anomaly detection requires significant computational capabilities, a robust dataset for training, and ongoing maintenance to ensure its efficacy in light of evolving anomalies and threats.

What is aimed to be achieved within the Thesis is to create and implement an anomaly detection system with usage of machine and deep learning.

Radford et al. proposed a unique model[3] in 2018. It analyzes the performance of five different sequence aggregation rules with the help of unsupervised anomaly detection techniques on the CICIDS2017 dataset[4]. Other studies based on signature-based detection usually use a frequency-based model. On the other hand, this model uses long short-term memory (LSTM) recurrent neural networks (RNNs) for modeling. Similarly, this research focuses on the identification of new or zero-day vulnerabilities in the context of unsupervised learning for identifying the unknown threats. It evaluates scores for each token in the sequence in view of the learned model and the subsequent tokens. Consequently, the study highlights the drawbacks of using the aggregated flow data to model the sequence and explains that the relative frequency may be more important than the sequence to detect attacks. It opens a new path for future research to apply deep learning directly on the packet level data.

The model by S. Garg et al. stands out due to the proposed hybrid model of ImGWO and ImCNN[5]. It aims to detect anomalies in real-time cloud network big data for data stream management systems. The suggested hybrid model uses Grey Wolf Optimization (GWO) for the feature extraction and Convolutional Neural Network(CNN) for the anomaly classification. Model GWO is enhanced by the ImGWO and CNN is enhanced by the ImCNN. This dual approach is to improve the feature selection and classification performance. Different from the conventional methods, this model transforms the data into RGB format for the ImCNN to process and it can handle large amounts of data and extract high-level features from the tcpdump logs. The study presents a contribution to network anomaly detection in the cloud environment.

The study of Siddiqui et al. (2019) is important through the application of unsupervised anomaly detection with explanations and expert feedback to improve the detection rate[6]. This approach focuses on the use of the Isolation Forest to identify anomalies in a data set that was gathered from over two million computers. It diverges compared to other methods as they produce good results only if there is a large set of labeled data. The novelty of this work comes from the application of Sequential Feature Explanation (SFE) to present explanations of the identified anomalies. It gives the security analysts an idea of which features are most impactful in the anomaly score. In addition, the work uses feedback from analysts who examine and categorize the anomalous cases. It helps to improve the model's

performance with the experts' assessment. Proposed method also offers an advantage of being able to learn by itself with little human intervention. This work differs from other studies because of the integration of anomaly detection, explanatory approaches, and the input from the subject-matter experts.

Nawir et al. (2019) worked on building a supervised machine learning system for network anomaly detection that focuses on minimizing the communication costs and network bandwidth[7]. Analyzing the UNSW-NB15 dataset[8], the research aims at establishing the most efficient algorithm based on the accuracy and time consumption. The most important contribution of this research work is the determination of the Averaged One Dependence Estimator (AODE) as the best algorithm that produced an accuracy of 97. 26 % and a processing time of approximately seven seconds. Experiments, performed in the WEKA environment, compared a number of supervised algorithms: Naïve Bayes, Multi-Layer Perceptron, Radial Basis Function Network, J48 Trees, and selected AODE as the best one. The proposed work also includes an investigation of the distributed version of the AODE algorithm to tackle the centralization problem in anomaly detection systems. This distributed approach is somewhat less precise (95. 86% and 96. 59%), yet it is efficient. This paper also demonstrates the compromise between the approaches' effectiveness. This work is distinctive in that it not only obtains good results on AODE but also introduces a distributed algorithm to overcome the problem of data centralization. This paper's strength in achieving accuracy, processing time, and network performance makes it a substantial contribution to the network anomaly detection field.

The study by Lin, Ye and Xu (2019) is a new approach to Network Security by using LSTM networks with an Attention Mechanism[9]. This system focuses on one of the most critical threats to computer networks namely the imbalances in the class distribution of the CSE-CIC-IDS2018 dataset[10]. Thus, when applying the SMOTE algorithm and an improved loss function, the study gets a very high accuracy of 96. 2% classification accuracy. This model is also evaluated based on the accuracy, precision and recall to show the model's efficiency as compared to other machine learning approaches. The deep learning model also uses TensorFlow and consists of two LSTM layers, three dense layers, and an attention mechanism to solve the problem of the time series of network traffic classification. This allows for the model to capture multivariate temporal dependencies of the data that is being

analyzed. The model applies these state-of-the-art methods effectively to overcome the problem of class imbalance and, thus, improve the identification of network anomalies. Paper states for future works, that it is possible to feed the neural networks with raw network traffic data to make the model learn the features on its own and enhance the model's performance.

Hwang et al. (2020) proposed the D-PACK[11], a system that is suitable for IoT environments and it incorporates CNN with an unsupervised deep learning model which is an Autoencoder. This model is especially effective for the early detection of anomalies since it only needs to inspect the first two packets of each flow; it offers almost perfect detection with a low false positive rate. This approach is useful especially due to the fact that IoT devices are among the most exposed to DDoS attacks, and the current protection mechanisms are insufficient. D-PACK is different by using auto-learning from raw data without the need to define the features beforehand. The system is proved to be efficient and scalable on datasets like USTC-TFC2016[12], Mirai-RGU[13], and Mirai-CCU. Mirai-CCU was built by the research team. The model can analyze large amounts of network traffic within a short time. This study becomes a new reference that can be used for early anomaly detection in IoT systems.

Lindemann et al. (2021) gives a detailed description of the application of LSTM networks in the identification of anomalies by creating a survey[14]. The paper identifies various types of anomalies. It compares LSTM-based solutions across various domains. It focuses on the architectures' capacity to identify perceptual anomalies because of their effectiveness in modeling temporal dependencies. It also discusses current developments such as graph-based and the transfer learning to capture the dynamics of the processes and the complex and heterogeneous data. A large part of the survey is devoted to the identification of the advantages of LSTM networks in time-series modeling and their use in different fields. The survey's findings suggest that future research should integrate LSTM networks with graph-based methods and transfer learning to improve detection performance and overcome the data heterogeneity issue. This paper is useful to get the overall idea of the existing and potential development of the LSTM based anomaly detection methods.

Ullah and Mahmoud (2021) addresses the problem of increasing the level of cybersecurity in the IoT systems[15] by their model. They glimpse into the utilization of CNNs to this end.

The model is tested on several datasets namely BoT-IoT[16][17] and IoT Network Intrusion[18] and proved to have high accuracy in detecting and differentiating different types of attacks. The model also adopts transfer learning for both the binary and multiclass classification, therefore demonstrating the model's flexibility when dealing with various IoT network traffic. This research is unique due to the proposed approach for anomaly detection based on CNNs and focusing on transfer learning to improve the model's performance. Despite impressive accuracy, precision, recall, and F1 scores, the paper also discusses possible issues in deploying the model in resource-limited IoT settings and the necessity to assess the model's performance across a more extensive dataset to counter new threats.

The study of Nassif et al. (2021) is a systematic review[19]. It makes the identification of multiple works related to anomaly detection based on machine learning. It divides the anomalies as point, contextual, and collective and also mentioned that the majority of the techniques belongs to unsupervised learning since they don't require labeled data. The review includes 43 types of applications, which proves the effectiveness of the use of machine learning in areas such as cyber security, industrial damage identification, and others. The review also stresses that the multi-dimensional performance metrics should be used for the evaluation of the models, and the phenomenon of the dominance of unsupervised anomaly detection is also discussed. Although the review is quite general and includes a vast number of studies, the author recommends performing new research using data sets from recent years and using various performance indicators to strengthen the conclusions towards the effectiveness and efficiency of machine learning models.

The paper by Sayed et al. (2022) proposes the use of LSTM networks and Autoencoders in identifying DDoS attacks in Software Defined Network (SDN)[20]. Using the techniques of Information Gain and Random Forest, the research work intends to predict the improvement in the performance of anomaly detection systems with less frequency of generating alarms. The evaluation of the proposed approach is carried out using three datasets, namely InSDN, CICIDS2017, and CICIDS2018. This research is differentiated from other existing literature in that it focuses on feature selection which is an important aspect in developing efficient anomaly detection models. LSTM and Autoencoders make the detection of the complex attacks easier and the incorporation of the two makes the model even better. This study shows that deep learning is capable of enhancing the security of a network especially in the

SDN environment.

The pair of Ullah and Mahmoud (2022) makes another contribution. The authors present a model that uses flow and control flag features in the identification of IoT network anomalies[21]. On datasets such as BoT-IoT and MQTT-IoT-IDS2020 it also delivers excellent results for binary and multiclass classification. The feed-forward neural network architecture is considered as efficient and accurate than the other models and hence it is best suited for the proposed model. This paper's key contribution is its new feature extraction method and the proper utilization of feed-forward neural network for anomaly detection in IoT networks. The consideration of the particular characteristics of the network traffic and the utilization of different datasets help to increase the model's reliability and its effectiveness in real-life conditions.

Hephzipah et al. (2023) suggests the new system enhanced by MMGT-ANN[22]. The KDD crime dataset has been used in the experiment, and the paper reveals that the discussed system outperforms others in terms of accuracy and time of cyber-crimes detection. The usage of MMGT for the selection of features and the fine-tuning of an ANN for predicting crime rates is a major contribution of this work. The described MMGT-ANN model is characterized by high accuracy and low time complexity, which makes it stand out from other models of cyber security. This paper's contribution in the feature selection and ANN optimization in enhancing the cyber security is therefore valuable for future work.

The paper of Wang et al. (2023) presents a comparative analysis of the most popular deep learning architectures, namely, DNN, CNN, RNN, LSTM[23]. In this study, the CSE-CIC-IDS2018 dataset is employed to stress on the significance of pre-processing and using several deep learning models for the enhancement of network intrusion detection. The specificity of this work is that a wide range of models was applied and special attention was paid to data preprocessing. Thus, the study presents the results of the comparison of several deep learning architectures and the impact of their usage on network security. The extensive effort invested in data preprocessing of a large dataset and the thorough comparison of various models are the major contributions of the research to the area of network anomaly detection.

1.2. Problem and importance of the problem

Detecting anomalies in computer networks is extremely important but also very difficult because of the large amount of complex network traffic. Networks produce huge quantities of data, which makes it hard to manually spot any unusual patterns or behaviors that could signal security risks, performance problems, or operational malfunctions. Conventional techniques like signature-based or rule-based detection have a tough time staying updated with the changing landscape of cyber threats, such as zero-day exploits and advanced persistent threats (APTs). These methods often lead to a lot of false alarms, which can overwhelm network administrators and potentially let actual threats slip through undetected. AI based approach improves in this aspect. However, detecting anomalies using artificial intelligence can be difficult. There is a need for a sufficient amount of accurate data that is correctly labeled for training the models. Getting these datasets can be challenging because network traffic data is usually extensive, varied, and may include confidential information. Also, machine learning models can be demanding computationally. They need significant resources for both training and ongoing analysis. Another challenge is the ever-changing nature of network environments and the evolving cyber threats. It means that constant updates and adjustments to the models are important for them to remain effective. Essentially, Computer Networks are important for modern digital society. They support different operations from personal communication to national security systems. Failing to detect and pointing out anomalies quickly can result in data breaches, financial losses and leaked sensitive information. It is important to improve anomaly detection capabilities to guard against cyberattacks and maintain network integrity. Implementing advanced methods like AI-based techniques is necessary to adapt to evolving threats and reduce the risk of unknown anomalies. They can enhance detection by identifying complex patterns and unknown threats that traditional methods may miss. Improving machine learning-based anomaly detection can lead to more accurate and efficient solutions. It can help reduce the risk of cyber-attacks and false alarms. This is essential for maintaining trust, security, and resilience in today's interconnected digital world.

1.3. Aim and importance of the study

In this thesis, a machine learning based system will be developed to detect anomalies in a computer network. There are many studies that use AI with flow-based anomaly detection.

However, trained models should be trained, tested and implemented to real world in order to keep trained models useful. Most of the offered systems within the literature works offline. They use pre-made datasets to train and test the models that are trained. They present results according to these datasets. Yet, there are very few works that try to implement trained models into live structure. In order for the studies to reach an impactful result, they should be implemented to the backbone of a corporation network or to an ISP structure. This implementation would result in thousands, even millions of flows to form. Waiting for this much flow from beginning to end and then evaluating it is not possible with the resources of these structures. Even when resources are increased, an increase will happen to traffic volume simultaneously, they will be overwhelmed within time. This situation requires a model to respond to a certain part of the flow immediately. What we aim for is being able to respond within this part of the structure.

Another aim of ours is for the model to make successful classifications by using the flowing traffic within the system. Thus, we are aiming to use NVIDIA Jetson AGX Orin and Nvidia Jetson developer kits which are kits that are specifically developed for deep learning implementation purposes.

1.4.Original contributions

In the literature review, two shortcomings were identified in the field of anomaly detection in computer networks. The first of these points is that the data sets used in the studies are outdated because they are not suitable for the changing network traffic structure and do not include new cyber threats and attacks. In Aldweesh et al. (2020), the authors state that only 5% of the data used in the studies consist of real-life or simulated data based on it. Roshan and Zafar (2021) make similar findings on the subject.

The second main deficiency in existing studies is that these studies are developed to work offline and are not directly applicable to real life. The reasons for not being suitable for real life are that the features used in traffic flows can usually be extracted after the entire flow is finalized, so the studies are carried out offline and the difficulties of implementing machine learning models in the embedded system.

This research intends to get results from the flow within a manageable time scale. In order for the system to be developed to be applicable to real life, different criteria will be used in

the selection and creation of flow characteristics. Finally, an embedded system will be used to support the machine learning model at the hardware level such as the backbone of the system.

1.5. Organization of the Thesis

The organization of the Thesis is as follows:

1.5.1. Chapter 1: Introduction and Background

This chapter contains introduction and related works dedicated to the IDS development. This chapter explains the problem, aim of the study and why is the intended contribution of the study.

1.5.2. Chapter 2: Materials and Methods

This chapter explains available materials for the development of the machine learning model and IDS system detection. It provides a detailed research methodology, Then, it provides how selected models were developed depending on available methods and materials. General structure of the selected methods was provided in here.

1.5.3. Chapter 3: Results

The results that are obtained by trained models and system are explained here. Results focus on distinction on selected models and datasets.

1.5.4. Chapter 4: Discussion

This chapter reveals an analysis of the results and their impact on IDS field. The potential of the provided system, its advantages over other systems and limitations of the study are discussed here.

1.5.5. Chapter 5: Conclusion and Future Work

The study is summarized here. The findings and contributions of the thesis to the IDS are wrapped up here. The impact of the study and its potential is outlined. This chapter ends with future recommendations to improve the system better with putting forward needing further validations of similar systems.

2. MATERIALS AND METHODS

This chapter outlines the materials used for developing the machine learning model and the intrusion detection system (IDS). It explains the research methodology in detail, and then describes how the selected models were built based on the available methods and resources. Additionally, the general structure of the chosen methods is explained here.

2.1. DATASETS

The datasets are an important element in making Intrusion Detection Systems (IDS) using Machine Learning better in network security development. These datasets help us train and test our ML models to detect and respond to cyber threats. This part discusses the important datasets used in IDS research, their evolution as well as their contributions and limitations.

The DARPA98 and DARPA99 datasets are among the first in IDS research. The DARPA and MIT Lincoln Lab collaboration created these datasets to simulate intrusions into military networks. DARPA99 built upon DARPA98 and added more types of attack. Although these datasets were foundational in their time, their realism was hindered by the fact that the traffic was synthetic. With this background as a basis, the KDD Cup 1999 (KDD99) [24][25] dataset came into existence and is widely used in the IDS field. KDD99, however, was not without shortcomings, such as redundant records and obsolete attacks.

To counter problems in KDD99 that were resulting in biased model validation, the NSL-KDD [26] was introduced in 2009. It solves the main problem of having redundant entries. NSL-KDD was better balanced and thus provided a fairer basis for evaluating IDS solutions that used ML.

The Kyoto 2006+ [27][28] dataset introduced the concept of using real network traffic captured over three years. The dataset classified the activity as normal, known attacks, and unknown attacks. It provided a long-term view of the network behaviour that shows what normal user behaviour would look like and what malicious user behaviour would look like. It used real traffic so it was much more realistic.

The MAWI Working Group Traffic Archive [29][30] also started capturing real traffic from the network. Instead of simulated attacks, the focus of this dataset was ordinary behavior. Nevertheless, it lacked attack labels, thus making it not quite useful for supervised learning, but actually useful to understand the normal behaviour of a network and anomaly detection.

Recently, the datasets from CAIDA [31] and DEFCON [32] gave samples of real network traffic that had both normal and malice activity. Though NSL-KDD and KDD99 had few modern attacks, newer datasets like UNSW-NB15[33] [34] [35] [36] [37] and CICIDS2017[4] improved on that with better attack features. The CSE-CIC-IDS2018[10] development further broadened the scope of attacks to include a more comprehensive and current range for IDS research.

2.1.1. DARPA98 and DARPA99 Datasets

The datasets of DARPA 1998 and 1999 Intrusion Detection Evaluation datasets were made available by DARPA to test on IDS performance. The Defense Advance Research Project Agency (DARPA) sponsored the datasets created in cooperation with MIT Lincoln Laboratories. Ultimately the created datasets aimed to provide benchmarks for IDS capabilities for detecting a long range of cyber-attack. Both the datasets seem to share some commonalities. However, they are quite different in nature. They also differ in terms of delayed scope. Moreover, the complexity and types of attacks simulated are also different. DARPA99 builds on the DARPA98 framework.

2.1.1.1.Content and Data Collection

The DARPA98 dataset has 7 weeks of traffic and audit logs. It was mostly UNIX-based, but also outsider attacks. The dataset was created to check the capabilities of the IDS through controlled normal and malicious traffic. The collected dataset contains a mix of benign and malicious network activities. It has numerous attack types which include Denial of Service (DoS), Remote to Local (R2L), User to Root (U2R), and Probing.

The DARPA99 data set is a follow-up to the DARPA98 data set and it increased the scope of the previous dataset by including Windows NT system in addition to UNIX systems. This version added an offline and real-time evaluation with the simulation of a network to capture real network activities. The environmental conditions concerning data collected were more

varied, including UNIX and Windows systems, meaning that we had a fairer assessment of the IDS.

2.1.1.2.Network Traffic Types

The DARPA98 dataset contained 38 individual attacks, both old and new. It helps test an IDS's ability to detect known and unknown intrusions. (22 words) There are four types of attacks in total:

- Denial of Service (DoS): Disrupts legitimate users' access to network resources, thereby rendering services unavailable.
- Remote to Local (R2L): Involves attempts to gain unauthorized access to a machine from a remote location.
- User to Root (U2R): A local, non-privileged user attempts to escalate privileges to gain superuser (root) access.
- Probing: Techniques used to gather information about the network, such as scanning ports or identifying vulnerabilities within the network infrastructure.

DARPA99 increased the attack capacity with 201 attack instances of around 56 attack types. Thus, it is more rigorous and less easily spoofed. This upgrade made sure that the IDS models can be tested in a setting offering a wider range of threats, such as newly introduced attacks and the modification of other.

2.1.1.3.Use in Research and Development

The DARPA98 dataset was a first in IDS research. It created the first comprehensive benchmark for intrusion detection methods. Scientists carried out research many times using DARPA98's datasets to develop IDS models that could detect attacks on UNIX based systems. They tested their approaches on a varied mix of network traffic.

DARPA99 helped IDS researchers with a more realistic environment for their evaluation, allowing them to test their systems effectively. This data led to the development of the KDD Cup 1999 (KDD99) which is heavily used in the IDS community. After KDD, the importance of the DARAP series in IDS technology development and evolution became insignificant.

2.1.1.4.Criticism and Limitations

Despite their foundational role, both DARPA98 and DARPA99 have been criticized for a number of limitations despite their foundational role. The datasets were based on **synthetic network traffic** generated in controlled environments. Synthetic datasets are generally artificial datasets where network traffic is built in controlled environments. Real-world network environments are not synthetic and involve more diverse and unpredictably complex traffic. The nature was fake and it leads to some kind of biases which may harm the performance of machine learning models.

In addition, DARPA98 and DARPA99 do not have any modern network protocols or current types of cyber threats. Which lowers the relevance of their evaluation. According to the DARPA99 report, the domain is more diverse than DARPA98. However, being a simulation, it has inherent limitations. These shortcomings made researchers look out of the box for newer datasets that are able to represent today's networks threats accurately.

2.1.1.5.Data Structure of the DARPA98 and DARPA99 Datasets

The DARPA98 and DARPA99 datasets have several common features, although DARPA99 expanded upon these:

- TCP/IP Packet Data: Both datasets contain packet-level data that is captured using TCP/IP protocol which includes packet headers and packet content in some cases. The information here tells like where is the source and destinations address, size of packets and kind of protocol used. This data gives important information used to evaluate networking events i.e. any interaction or communication between the devices. This can be data transfers, attempts to connect, disconnection of sessions or anything of like.
- Time Stamps: Each network event in the datasets is associated with a timestamp, which is crucial for temporal analysis of network behavior. This allows researchers to track changes in activity over time and identify potential attack patterns.
- Labels: The labels are assigned to each network event signifying whether it is normal or an attack of specific type. These labels form the basis of supervised learning which helps train machine learning models to detect various intrusion types.

- Additional Features: The DARPA99 dataset includes additional features that provide insights into network flows, session statistics, and other network-level behaviors. This provides insights into network flows, session statistics, and other network-level behaviors. Hence, it is better than DARPA98.
- Data Format: Both dataset's data is formatted in tcpdump-compatible formats so network analysis tool data. This way they can be used by researchers for various types of IDS testing and evaluation.

2.1.1.6.Simulation Environment

The DARPA98 dataset was generated in a simulated environment that closely resembled real conditions, albeit simplified. The network contained several UNIX machines that mimicked a U.S. An Air Force local-area network which provides a suitable testbed for evaluation of IDS capabilities. Yet, because the dataset was synthetic, it limited the randomness and variability present in real networks.

The original simulation environment was modified to include the Windows NT system, in addition to UNIX, to form a more realistic mixed-OS environment by various research groups. Even with these advancements, the setting remained artificial, meaning the dataset was not real.

The datasets of DARPA98 and DARPA99 are important to help in the production of IDS systems. The researchers tested their models on the dataset and then worked on improving them, and their impact played a direct role in the **KDDCup99** dataset as cyber threats and network environments evolve, it is important to have modern, realistic datasets that can depict a network security scenario more faithfully.

2.1.2. KDDCup99 Dataset

The KDDCup99 dataset is a very popular dataset that has been used as an evaluation of any Intrusion Detection System. The dataset presented in this paper was created for The Third International Knowledge Discovery and Data Mining Tools Competition (KDD-99) which was held in conjunction with the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. It is a benchmark dataset for IDS research. KDDCup99 is used for IDS model development and benchmarking, it is derived from

DARPA98, it is a dataset of normal traffic and malicious traffic. KDDCup99 aimed to provide a collection of network traffic data, including both normal and malicious activities, to support the development and evaluation of IDS models.

2.1.2.1.Content and Data Collection

The MIT Lincoln Laboratory did a special program called the DARPA98 IDS in 1998. From this program, the KDDCup99 dataset was created. It was developed to provide researchers working on IDS with a general-purpose database. The dataset contains normal network activities and numerous attacks. Therefore, it provides a bedrock for future researchers to compare various IDS models. To create the dataset, a wide variety of attack types were used in order to capture the intrusion behaviour of various attacks.

It is a much simpler and preprocessed version of DARPA 1998 dataset. The dataset is presented as a set of connection records. Each instance is described by 41 features (e.g. duration, protocol type, service, number of bytes transferred, etc…). Each connection being labelled as either normal or an attack which is one of 39 types. User friendly for machine learning to train and test easily.

The dataset led to the creation of other better datasets, like the **NSL-KDD**, which fixed some flaws of the KDDCup99. KDDCup99 has inspired a large volume of research work on intrusion detection systems, although still exhibiting some flaws after over 20 years.

2.1.2.2.Criticism and Limitations

Although widely employed, the KDDCup99 dataset has many limitations that previous researchers have pointed out:

- Redundancy: The dataset includes several duplicate entries that can cause bias and affect the performance of machine learning models. When data records are repetitive, the models get influenced easily. As a result, they become unable to differentiate between attacks which haven't been seen before.
- Lack of Temporal Information: KDDCup99 lacks temporal and sequential data, which are crucial for understanding the behavior of attacks over time. This absence limits its applicability for evaluating IDSs that rely on sequential or time-series analysis.

- Unrealistic Data Distribution: The distribution of normal and attack instances in the dataset does not reflect real-world network environments. The dataset has an overrepresentation of some types of attacks, which may not be typical of actual network traffic patterns.

- Outdated: The dataset is derived from DARPA98The dataset of Darpa98 is over twenty years old. Thus, it does not take into account attacks that occurred after this date, nor does it take into account the network protocols that exist today.

Because of these issues, newer datasets have come up, like the NSL-KDD dataset, which tries to solve this issue by removing redundant records and providing a better balanced dataset.

2.1.2.3.Simulation Environment

The KDDCup99 dataset was generated based on the DARPA98 evaluation, which means that it was collected in a **simulated network environment**. The simulation took place with a controlled network activity and attack situation with the evaluation of the IDS to check its capabilities to detect the previously known attack. Though a wide range of attacks are simulated, it does not encompass the high variability and unpredictability of the real-world network environment.

2.1.3. NSL-KDD Dataset

The NSL-KDD dataset is an upgraded version of the KDDCup99 dataset. This fix was made to produce the limitations of KDDCup99. Created by the Canadian Institute for Cybersecurity at the University of New Brunswick, NSL-KDD has become an important benchmark for Intrusion Detection System (IDS) research, especially in the field of machine learning. NSL-KDD aimed to provide a better dataset for intrusion detection systems by fixing the weaknesses of the KDDCup99 dataset.

2.1.3.1.Improvements Over KDDCup99

The NSL-KDD data set was generated to tackle some of the most important issues of KDDCup99. Below are the notable improvements that distinguish NSL-KDD:

- Removal of Redundant Records: The NSL-KDD dataset does not have duplicate records and irrelevant records which were in abundance in KDDCup99. The KDD Cup 99 dataset has a lot of duplicates. As a result, our model biases. Thus, it has become overly tuned to frequent records. Therefore, it is less generalizable. NSL-KDD was used to tackle this problem by ensuring that this dataset has lesser redundant records thus, leveling the dataset.

- Balanced Dataset: The training and testing sets of NSL-KDD were adjusted to ensure similar record sizes, which helped prevent overfitting. By maintaining a more balanced representation between normal and attack records, NSL-KDD ensured that machine learning models trained on this dataset were not disproportionately influenced by particular types of attacks.

- No Need for Random Subsampling: The KDDCup99 dataset is very large. Researchers have to perform random subsampling because it is too large for experimentation. The approach led to a missing partial or biased representation of network activities. NSL-KDD tackled this problem by offering a more realistic quantity of data, which can be utilized directly in the training and testing of IDS models without arbitrary reduction.

2.1.3.2.Content and Data Collection

Much like KDDCup99, the NSL-KDD dataset represents network connections as a vector of features. The dataset retains the structure of KDDCup99 by providing 41 features for each connection, which include basic attributes of TCP connections, content-based features, and traffic characteristics derived from a window of network connections. Each connection is labeled. These labels provide a strong foundation for supervised learning approaches in IDS research. NSL-KDD essentially captures a range of attack scenarios similar to KDDCup99, but with the enhanced balance and reduced redundancy discussed above.

2.1.3.3.Limitations

Despite the improvements introduced with NSL-KDD, it still has limitations that prevent it from being fully representative of modern network environments:

- Outdated Data: NSL-KDD is derived from KDDCup99, which, in turn, is based on the DARPA98 data collected more than two decades ago. As such, it lacks coverage of more recent types of attacks, as well as current network technologies and protocols. The outdated nature of the data reduces its applicability in testing IDS models meant for today's rapidly evolving cyber threat landscape.

- Retains Basic Features and Format of KDDCup99: Although NSL-KDD addressed some KDDCup99 problems, such as balancing and redundancy removal, it is still based on KDDCup99's feature set and basic connection format. Modern network traffic and attack patterns cannot be completely captured with these characteristics.

2.1.3.4.Use in Research and Development

Widely considered an important new step in the development of IDS datasets, NSL-KDD set. It has kept a range of KDDCup99 strengths at the same time it has eliminated any major faults. Consequently, the NSL-KDD dataset has been extensively used to benchmark the performance of IDS models based on machine learning. The more balanced and manageable dataset have enabled researchers to train and evaluate an IDS model with less bias and better generalization than KDDCup99.

NSL-KDD is more refined version of KDD dataset which might not suit the researchers however still has value. However, owing to the sophistication of cyber threats and the evolution of network technology, researchers have been increasingly looking for newer datasets.

2.1.3.5.Summary and Evolution of Datasets

The progression from DARPA98 to KDDCup99, and subsequently to NSL-KDD, illustrates a continuous effort to address the limitations of earlier datasets and to improve the quality of IDS training data:

- DARPA98 and DARPA99: Foundational but lacked the realism of real-world network conditions.
- KDDCup99: Expanded the accessibility and usability for machine learning but introduced biases and redundancy.
- NSL-KDD: Removed redundant records, provided a more balanced dataset, and minimized the need for random subsampling, thus improving the dataset's practicality for IDS research.

However, given that the data for NSL-KDD still originated from a period when network technologies and threats were vastly different from today, its usability is limited. The need for more recent and representative datasets has spurred the creation of newer benchmarks like UNSW-NB15 and CICIDS2017, which attempt to better capture the complexities of network environments and the evolving landscape of cybersecurity threats.

Following **Table 2.1** given below describes the four datasets on network security and how they evolved over a period of time. It also shows advancements made in each of them.

Table 2-1 Summary of the differences between the DARPA98, DARPA99, KDDCup99, and NSL-KDD datasets

Feature/Aspect	DARPA98	DARPA99	KDDCup99	NSL-KDD
Year of Creation	1998	1999	1999	2009
Content	Simulated network traffic, including various types of cyber attacks and normal activities.	Simulated network traffic with more diverse attack scenarios and data collection.	Processed set of attacks and normal traffic with labeled features.	Processed set with redundant and duplicate records removed, and a balanced dataset with refined training and testing splits.
Types of Attacks	DoS, R2L, U2R, and Probing.	DoS, R2L, U2R, and Probing with more diverse instances and multi-stage scenarios.	DoS, R2L, U2R, and Probing.	DoS, R2L, U2R, and Probing with improved representation and balance.
Number of Attacks	32 unique attack types.	58 unique attack types.	39 attack types derived from DARPA99.	37 attack types with improved distribution and reduced redundancy.
Data Format	Raw network traffic data in tcpdump format, requiring pre-processing.	Raw network traffic data in tcpdump format with additional traffic diversity.	Processed features (41 features per connection, labeled) in CSV format.	Processed features in CSV format, with improved data formatting and reduced redundancy.
Labeling	Not labeled; required processing and manual annotation.	Not labeled; required manual processing for analysis.	Labeled as normal or attack with specific attack types.	Labeled with reduced redundancy and balanced distribution.
Limitations	Lack of realism, limited attack types, and artificial traffic patterns.	Limited realism and representativeness, scalability issues.	Redundant records, duplicate data, lack of temporal context, biases.	Based on outdated traffic, may not represent current network scenarios and attack vectors.

2.1.4. UNSW-NB15 Dataset

The UNSW-NB15 dataset is a comprehensive modern dataset developed for the evaluation of Network Intrusion Detection Systems (NIDS). It aims at overcoming the shortcomings of previous datasets, like KDDCup99 and NSL-KDD. In 2015 was developed by the Australian Centre for Cyber Security (ACCS) at the University of New South Wales, it offers a realistic testing ground for IDS research in the context of the latest cyber threats. By capturing normal and malicious activities that reflect the real world, UNSW-NB15 uniform dataset offers great potential value to improve the performance of IDS technologies.

2.1.4.1.Content and Features

The UNSW-NB15 dataset consists of a large amount of network traffic data representing benign and attack behavior. It has **49 features** relevant to the network communications. These features include:

- General Network Flow Features: This can include the source and destination IP addresses, port numbers, and protocol types, which give basic information about individual network connections.

- Content-Based Features: It includes the details of the data transferred in a connection. Useful for identifying payload-based attacks.

- Calculated Traffic Features: Traffic features based on differential monitoring over a window of connections. this feature would lend insights into traffic patterns of the network. this feature would be useful in indicating some kind of anomaly.

This mix of features makes UNSW-NB15 suitable for comprehensive intrusion analysis, allowing the detection of complex patterns and behaviors that may indicate an intrusion.

2.1.4.2.Types of Attacks

UNSW-NB15 stands out from previous benchmarks through its coverage of a broad spectrum of contemporary cyber threats and thus, it provides a more comprehensive view of modern attack scenarios. There are nine types of attacks in the dataset:

- Shellcode: It refers to malicious in nature, with the aim to compromise a system and acquire control over it.

- Fuzzers: Attacks that send unexpected or random input to systems to discover vulnerabilities.
- Analysis: Analysis is the malicious activity aimed at gathering information or analyzing vulnerabilities, often part of a reconnaissance campaign.
- Backdoors: Backdoors are a way of covertly getting remote access. It is usually installed by the Hacker after an intrusion.
- Denial of Service (DoS): They aim to make a network service unavailable to users. This is done by flooding the service with requests so that it can't handle it and will either crash or become unusable.
- Exploits: Exploits are attacks that make use of specific code vulnerabilities in software or systems via unauthorized actions.
- Generic: Attacks that are not tied to a specific platform or software but exploit common network or protocol vulnerabilities.
- Reconnaissance: Methods like scanning to gather information about the network and identify weaknesses.
- Worms: Worms are self-replicating malware that spread in a network. They are primarily used to create botnets or compromise a large number of hosts.

The assorted and numerous types of attacks contained in UNSW-NB15 are comparable to the current threats out there, making it a useful benchmark for evaluating IDS model simulation.

2.1.4.3.Data Collection

The IXIA PerfectStorm tool was used to generate the UNSW-NB15 dataset that simulates real attack behaviour as well as normal network traffic. The process of data collection included capturing:

- Real Normal Behaviors: Traffic generated to mimic legitimate user activities, ensuring that the dataset reflects typical network operations.
- Synthetic Attack Traffic: Created to represent contemporary attack techniques, providing a realistic context in which to evaluate IDS performance.

UNSW-NB15's realistic dataset and unsophisticated simulated attacks provide a high degree of accuracy for more modern IDS technologies. Thus, we can conclude that these activities can be used as a reliable and relevant dataset.

2.1.4.4.Use in Research and Development

UNSW-NB15 data is widely accepted and used in both academic and industrial research for assessing IDS models, especially ones based on machine learning and data mining techniques. The dataset is realistic of the currently occurring traffic and incorporates newer attacks. Therefore, it is suitable for the evaluation of the intrusion detection model compared to the older datasets like KDDCup99 and NSLKDD.

The dataset has many features, so researchers can try different approaches which range from simple rule-based systems to complex deep learning systems. The balanced normal and attack data included allows assessment of model performance in identifying an intrusions with negligible false positive rates amidst modern cyber threats.

2.1.4.5.Criticism and Limitations

UNSW-NB15 does have its limitations despite being an improvement over the previous datasets. Some of these limitations are:

- Modeling Real-World Environments: While UNSW-NB15 captures a wide range of attack types and mimics real traffic patterns, it is still collected in a controlled, synthetic environment. As a result, it may not fully replicate the chaotic and unpredictable nature of actual production network environments. Researchers using this dataset need to be mindful that real-world networks are often messier and contain traffic types that are not covered in the dataset.

- Dependence on Simulation Tools: The reliance on tools like IXIA PerfectStorm means that attack scenarios, while realistic, are limited by the capabilities of the tool. The dataset may not include all possible forms of network intrusions, especially highly sophisticated attacks that leverage advanced obfuscation techniques or rapidly evolving attack vectors.

2.1.5. MAWI Working Group Traffic Archive

The MAWI Working Group Traffic Archive is a key data set for network research. The Japanese academic backbone network Widely Integrated Distributed Environment was created and runs the MAWI Working Group; measurement and analysis on the WIDE Internet (MAWI), MAWI data set. This dataset has been around since 1999 and has given researchers actual Internet Traffic Data that greatly improved the understanding of the network behaviour, performance and security.

2.1.5.1.Content and Data Collection

The goal of the MAWI Traffic Archive is to provide researchers with a source of real data on the behavior of network traffic. Various points at the backbone of WIDE Project which connects different research and educational institutions in Japan collects Data. This data cover a vast geography and give wide perspective on the network activity.

The way that data is collected is using methods like a packet sniffer and other monitoring tools to capture IP packets and, in some cases, application-level data. To maintain privacy, we anonymize the data we collect by masking IP addresses and other identifying information. The MAWI archive is specifically designed to ensure that the data being used is useful to researchers and protects the privacy of the participants involved.

The dataset provides us with two types of information:

- Packet Traces: These are raw captures of network traffic, like IP headers, and sometimes payload information. This information shows us how specific things connect.

- Flow Data: The summary of packets consists of flows that embody common features, for instance, source address and destination address or source port and destination port and so on. This kind of data helps in spotting high-level traffic patterns and behaviors.

2.1.5.2.Network Traffic Types

The MAWI Working Group Traffic Archive has a date of benign and malicious data. The dataset has events and activities that happen every day as well as strange or suspicious activities. This dataset is very useful for researchers as they can use it for doing anything with the network. For instance, it can help researchers find out about normal network usage or malicious usage.

2.1.5.3.Applications in Research and Development

The MAWI Traffic Archive has numerous applications within network research and education:

- Traffic Flow Analysis: When researchers look at network flow patterns, they can tell what things are common and what's uncommon. This may help them better understand how resilient common protocols and architectures are.
- Network Security Assessment: The archive assists identifies normal and abnormal events to study attack detection, intrusion prevention, and related cybersecurity domains.
- Performance Evaluation: It ensures a proper setting of the network under different traffic conditions and protocols.
- Educational Use: The archive also supports educational use by providing students with access to realistic network traffic for analysis. By working with live network data, students will get the opportunity to bridge the gap between theory and practice.

2.1.5.4.Accessibility and Data Format

A key feature of the MAWI Traffic Archive is that it is publicly available. The MAWI Working Group offers the dataset in common formats (pcap) compatible with popular network analysis tools (e.g., Wireshark, Tcpdump). Researchers, educators, and students can easily access the MAWI archive. Nonetheless, it should be noted that most of the archived files are unlabeled which may require heavy pre-processing for some applications like training up a supervised learning model for intrusion detection.

2.1.5.5.Limitations and Privacy Considerations

Despite its value, the MAWI Traffic Archive has some limitations. The data is unlabeled, making it challenging to use directly for supervised machine learning applications without extensive preprocessing. Furthermore, while the data provides real-world traffic, privacy concerns require that sensitive data be anonymized, which may result in a loss of some useful network-level details. Nevertheless, the MAWI archive maintains a strong balance between data usability and participant privacy, making it an invaluable resource in the study of real network environments. It has played an irreplaceable role in the development of network research by providing real, high-quality network traffic data for analysis. This data complements theoretical work and simulations, ultimately leading to more realistic and effective solutions in network security and technology.

2.1.6. CICIDS2017 Dataset

The CICIDS2017 data set made by Canadian Institute for Cybersecurity for the evaluation of network Intrusion Detection Systems (IDS). This dataset was created in order to overcome the shortcomings of older datasets like KDDCup99 and NSL-KDD by giving IDS researchers a more complete, realistic and recent dataset. Since it was developed, the CICIDS2017 dataset has established itself as a benchmark of comparison for network IDS models.

2.1.6.1.Content and Data Collection

The goal of the CICIDS2017 dataset was to mimic legitimate network traffic including the benign ones and the different kinds of attacks. The data was collected over multiple days and represented in multiple CSV files. Each file represents a different day of capture or type of network usage.

The dataset emulates real-world network environments, comprising normal traffic along with a variety of attacks like DDoS, Heartbleed, Botnet, Infiltration and different types of web attacks. The dataset is diverse enough to help in training models for different types of network anomalies and malicious behaviours.

2.1.6.2.Network Traffic Types

The CICIDS2017 dataset has a variety of **normal and malicious** network traffic activities. The dataset contains **15 labels** that consist of the attacks and normal traffic. These labels are:

1. BENIGN: Normal network activity without any malicious behavior.
2. DoS Hulk: A Denial of Service (DoS) attack that floods the target with large amounts of data to disrupt normal services.
3. PortScan: A reconnaissance attack where the attacker scans various ports to identify vulnerable services running on the target machine.
4. DDoS: A Distributed Denial of Service attack, which involves multiple systems flooding a target to render it unusable.
5. DoS GoldenEye: A specific DoS attack designed to overload a web server by sending a large number of HTTP requests.
6. FTP-Patator: A brute force attack aimed at the FTP service to gain unauthorized access by attempting numerous login credentials.
7. SSH-Patator: Similar to FTP-Patator, this attack attempts to brute force access to the SSH service.
8. DoS Slowloris: A type of DoS attack that tries to keep connections open with the target web server as long as possible, thus preventing legitimate requests from being fulfilled.
9. DoS Slowhttptest: Another DoS attack that sends HTTP traffic at a very slow rate, attempting to exhaust server resources.
10. Bot: This attack involves the use of malware to infect and take control of devices, allowing attackers to conduct further attacks.

11. Web Attack - Brute Force: A web attack that uses brute force techniques to try different combinations of login credentials to gain unauthorized access.

12. Web Attack - XSS (Cross-Site Scripting): An attack targeting web applications to execute malicious scripts in the user's browser by exploiting vulnerabilities in web pages.

13. Infiltration: This attack involves unauthorized access to internal networks, often by exploiting vulnerabilities in network defenses.

14. Web Attack - SQL Injection: A web attack that involves inserting malicious SQL queries into input fields to manipulate the backend database.

15. Heartbleed: A vulnerability in the OpenSSL library that allows attackers to read sensitive data from the memory of web servers, including encryption keys and passwords.

This broad coverage of attack vectors allows for thorough evaluation of IDS models, testing their ability to distinguish **benign traffic** from a diverse array of **malicious behaviors**. It also enables detailed multiclass classification tasks, as models can be trained to identify each specific type of attack.

2.1.6.3.Features and Data Format

Each network flow was described by extracting **84 flow-level features** from the **CICFlowMeter** tool that depicts the traffic. The features describing each flow were designed for training of the machine learning model. Several types can be seen in features. The features can be categorized as follows:

- Basic Flow Features: Essential information like source and destination IP addresses, port numbers, protocol type as well as a timestamp.

- Packet-Level Features: This type includes information on the length of each packet, the number of packets, the flags, etc. Noise and other related features help in understanding the makeup of data transmitted over the network..

- Flow Statistics: Statistical features like flow duration, number of packets per second and bytes per second describes the behaviour of a whole network flow.

- Forward (Fwd) and Backward (Bwd) Inter-Arrival Times: These features use timing packets in forward and backward directions, for example total inter-arrival time and total average inter-arrival time.

- Flow Active and Idle Times: The details include information about the times the stream was active or idle. They aid in discovering the spikes in your network activity, and idle times.

- Flow-based Timing Features: This category gives information about flow times, their mean, maximum, minimum and standard deviation, so that time-based properties of network activity can be analyzed.

- TCP Flag Features: The packet captures show TCP metrics like SYN, ACK, FIN, other TCP flags, etc., to understand what is being established.

- Additional Packet Count Features: The features of this category are recorded at a packet level. For example, the number of packets sent in the forward or backward direction per second.

- Subflow Features: Subflow features can break network flows into smaller pieces that add more information about packets and bytes in subflows in the forward and backward directions.

- Window and Segment Features: The initial window sizes and minimum segments sizes seen in the forward and backwards directions are features to put some light on congestion control issues.

- Label: Each network flow is labeled as either benign or malicious, so it is a great dataset for supervised learning.

Packet, flow and network level insights are available in **CSV** format. The format is easy to process and can be directly used in machine learning.

2.1.6.4.Labeling and Classification

Each instance of a network flow in the dataset is labeled; either benign, or one of the attack classes. Due to the classification of the input data, it is suitable for supervised learning. In other words, the model should learn to detect whether the network event is normal or malicious.

The labels in the dataset also identify the various attacks that enable the models for multiclass classification that help distinguish between the DoS, DDoS, Infiltration and Web Attacks. The extensive labeling allows for a detailed ground truth on which the researcher can test and validate the effectiveness of IDSs in identifying wide and subtle attack behaviors.

2.1.6.5.Use in Research and Development

The CICIDS2017 dataset has been widely used by researchers to develop and evaluate IDS models. Its realistic portrayal of both normal and abnormal network behavior provides a solid foundation for:

- Supervised Learning: The dataset is useful for training machine learning models that aim to distinguish between normal traffic and malicious behaviors.
- Anomaly Detection: The rich feature set enables models to learn to detect anomalous patterns that could indicate emerging threats or zero-day attacks.
- Attack Pattern Analysis: The diversity of attacks provides a basis for evaluating how effectively different IDS models can identify specific attack types.

2.1.6.6.Limitations

Although this set is helpful, CICIDS2017 dataset has some drawbacks:

- **Imbalanced Data**: Models that are used for attack detection sometimes do not have enough data on a certain attack type as compared to other attack types.
- **Lack of Novel Attacks**: Although there is a variety of sample data, it still does not contain every emerging attack type. Therefore, this can limit its suitability for the evaluation of IDSs. In addition, it can be against the most recent types of attacks.
- **Preprocessing Requirement**: Processing the specification might take long for systems with high number of complex features. Techniques performed to a dataset before feeding it into the model is called Preprocessing. It entails **redundancy, handling missing value and normalizing** features.

2.1.6.7.Conclusion

The CICIDS2017 dataset is a significant contribution to IDS research. It is helpful because it provides realistic network traffic and multiple attack scenarios, and serves to fill in many of the gaps in datasets like KDDCup99 and NSL-KDD. The data set has many features which are highly suited for the classification as well as the anomaly detection task.

Yet, as cyber threats and networking technologies are evolving through time, there is a requirement for more up-to-date datasets to keep repeat up to date data. Nevertheless, **CICIDS2017** still ranks amongst one of the most recent ones that help train and benchmark IDSs. This is particularly the case for those working on enhancing the detection and counteraction of modern attacks that happen over computer networks.

2.1.6.8.CICIDS2018 Dataset

The CICIDS2018 dataset is an extension of the CICIDS2017 dataset, both of which were developed by the Canadian Institute for Cybersecurity (CIC). The main differences between the two datasets are:

1. Data Collection Duration:
    - CICIDS2018 was collected over ten days (February 14th to March 2nd, 2018), whereas CICIDS2017 was collected over five days (from July 3rd to July 7th, 2017). The longer collection period of CICIDS2018 aims to capture more data but doesn't necessarily translate into greater coverage or representativeness.

2. Attack Scenarios:
    - CICIDS2018 contains additional attack types such as insider threats and data exfiltration, alongside common attacks like DDoS and brute force. CICIDS2017 also features diverse attack scenarios, including Web Attacks, Heartbleed, Infiltration, and DDoS, which effectively represent a wide range of network threats. The differences in attack types do not necessarily indicate superiority but rather reflect variations in threat modeling approaches between the datasets.

3. Feature Extraction:
    - Both datasets contain 84 features from CICFlowMeter. While CICIDS2017 uses older version, CICIDS2018 uses CICFlowMeter-V3. The features that were taken out are more or less the same, no new type or usefulness of features.

4. Purpose and Applicability:
    - Both datasets contain 84 features from CICFlowMeter. While CICIDS2017 uses older version, CICIDS2018 uses CICFlowMeter-V3. The features that were taken out are more or less the same, no new type or usefulness of features.

Both CICIDS2017 and CICIDS2018 datasets are useful to evaluate intrusion detection systems. Although the time span, attack scenarios, and data collection methods differ the two datasets can't be said to be better than the other. It depends on your intrusion detection research but each of them can suit your needs. Response It is a benchmark data collection for anomaly detection and other studies.

2.1.7. Comparison of the Datasets

It is important to have a comparison of the available datasets to have a better understanding. Following **Table 2.2** shows a comparative overview of the discussed datasets so far.

Table 2-2 Overview of Datasets for Intrusion Detection Systems

This table shows a comparison between datasets used in IDS training systems.

Dataset	Year of Creation	Data Collection Type	Number of Attack Groups	Features	Duration of Collection	Labeling	Key Improvements	Limitations
DARPA98	1998	Simulated Network Traffic	4	Raw Packet Data	7 weeks	Normal/Attack Label	Foundational dataset for IDS	Synthetic traffic, outdated attacks
DARPA99	1999	Simulated Network Traffic	4	Raw Packet Data	5 weeks (expanded to real-time evaluation)	Normal/Attack Label	Expanded attack types, included Windows NT systems	Synthetic traffic, outdated attacks
KDDCup99	1999	Derived from DARPA98	4	41 Features	Derived from DARPA98	Normal/Attack Label	Simplified data for ML, large-scale	Redundant entries, unrealistic distribution, outdated
NSL-KDD	2009	Improved from KDDCup99	4	41 Features	Derived from KDDCup99	Normal/Attack Label	Removed redundancy, balanced dataset	Still based on outdated data, lacks temporal information
UNSW-NB15	2015	Real & Synthetic Traffic	9	49 Features	IXIA PerfectStorm Tool	Normal/Attack Label	Realistic traffic, nine modern attack types	Limited ability to fully model real-world environments
MAWI Archive	Since 1999	Real Network Traffic	-	Packet and Flow Data	Ongoing	No labels	Real-world, large-scale traffic	No specific labels, lacks systematic attack modeling
CICIDS2017	2017	Simulated Realistic Traffic	15	84 Features	5 days	Normal/Specific Attack Label	Diverse attack scenarios, supervised learning	Imbalance between attack classes, limited timeframe
CICIDS2018	2018	Simulated Realistic Traffic	20	84 Features	10 days	Normal/Specific Attack Label	Longer data collection, additional attacks	Imbalance between attack classes, similar limitations as CICIDS2017

2.2.METHODOLOGY

Methodology used for model training and system is explained in this section of the thesis.

2.2.1. Dataset Acquisition

This research employs the CICIDS2017 dataset, a common benchmark for network intrusion detection system (IDS) research. (21 words) The CICIDS2017 dataset created by the Canadian Institute for Cybersecurity simulates real-life threats with different activities that include normal traffic as well as different attacks. this diversity makes it well appropriate for IDS evaluation. You can have labeled flow data, raw network captures (PCAPs), and pre-processed CSV files for research usage. For this research, the MachineLearningCSV version was chosen, as it can be directly used for machine learning and required lesser preprocessing. We downloaded the data using google colab because it has enough power and it easily integrates with the google drive.

2.2.1.1.Data Cleaning and Label Normalization

After unzipping the dataset, all the csv files were checked for issues like wrong labels and corrupt data. It was noticed that in one of the files, Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv, there are some characters that were not recognized in the attack labels with which the web attacks are named. Hence there was a lack of understanding of what type of attack was represented in (For instance Web Attack Brute Force). These symbols which were previously not identified were modified so that all of the characters in this data are made uniform to aid machine learning algorithms. The unknown characters were replaced with standard characters to maintain consistency in naming. For example:

- "Web Attack Brute Force" was renamed to "Web Attack-Brute Force"
- "Web Attack XSS" was renamed to "Web Attack-XSS"
- "Web Attack Sql Injection" was renamed to "Web Attack-Sql Injection"

Cleaning the Data was important so that there are no issues while training the models due to the different labels assigned to different datasets.

Table 2.3 provides an overview of the extracted files:

Table 2-3 Overview of Extracted Files from the Dataset

File Name	Description
Monday-WorkingHours.pcap_ISCX.csv	Network traffic from Monday, including benign data
Tuesday-WorkingHours.pcap_ISCX.csv	Network traffic from Tuesday, including benign data
Wednesday-workingHours.pcap_ISCX.csv	Network traffic from Wednesday, including benign data
Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv	Traffic from Thursday morning, including web attacks
Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv	Traffic from Thursday afternoon, including infiltration attempts
Friday-WorkingHours-Morning.pcap_ISCX.csv	Traffic from Friday morning, including benign data
Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv	Traffic from Friday afternoon, including port scanning attacks
Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv	Traffic from Friday afternoon, including DDoS attacks

2.2.1.2.Combining the Dataset Files

All the CSV files were cleaned and labels normalized after which all of them were made into one CSV file that would be used to train the model. This compilation made it possible to analyze the whole network activity including the normal (benign) behavior as well as the attacks. Merging the files enabled shuffling and splitting of the dataset during the model training and evaluation phases, effectively.

The combined process involved reading each of the csv files and merging them in one DataFrame. This gave us a very rich dataset which includes many aspects of network activity: benign traffic and attack. The purpose was to provide the machine learning model with a varied training dataset that closely represented actual network conditions.

2.2.2. Data Integrity and Feature Check

Once the CSV files were merged, a count of the various class labels was carried out to check the data integrity and estimate the distribution of different labels of dataset. The dataset consisting of Normal and Attack records can be seen in **Table 2.4**:

Table 2-4 Distribution of Traffic Types in the CICIDS2017 Dataset

Traffic Type	Record Count
BENIGN	2,273,097
DoS Hulk	231,073
PortScan	158,930
DDoS	128,027
DoS GoldenEye	10,293
FTP-Patator	7,938
SSH-Patator	5,897
DoS slowloris	5,796
DoS Slowhttptest	5,499
Bot	1,966
Web Attack - Brute Force	1,507
Web Attack - XSS	652
Infiltration	36
Web Attack - SQL Injection	21
Heartbleed	11

This thorough assessment of the distribution of labels showed that the data will provide a wide variety of traffic scenario, which is crucial for training the IDS to detect the frequent as well as the rare types of attacks. It was remarked that certain types of attacks like Heartbleed and SQL Injection didn't happen as often as others. We must consider this skewness at the time of model training to avoid bias.

The dataset was cleaned and merged before saving to Google Drive for the next stage model training. This tactic allows them to save the data in a clean and formatted way, along with a checkpoint for running the same experiments again or making changes without repeating data preparation. The dataset was then saved as **MLCVE.csv** and also backed up at Google Drive folder.

2.2.3. Dataset Preprocessing Variants

After getting the original MachineLearning version of the CICIDS2017, we had to perform many preprocessing to get a more balanced dataset for model training. This section focuses on done preprocessing on original dataset and subsequent variants formed of these changes.

2.2.3.1.Overview of Preprocessing Techniques

To prepare the CICIDS2017 dataset for model training, this research employed multiple preprocessing techniques. Different variations of preprocessing were used to show how data, if transformed, affects accuracy, generalizability and performance. In this section, we will focus on the first dataset called MLCVE dataset which had almost no pre-processing as most of the features from MachineLearningcvs version of CICIDS2017 dataset were retained.

2.2.3.2.Dataset 1: The CICIDS2017 Machine Learning Dataset (Original Dataset)

The MLCVE Dataset used in this research is not a variant created for this study but is instead the **MachineLearningCVS.zip** file from the CICIDS2017 dataset. We used this dataset as a baseline in its original form to check for further preprocessing before moving on to training machine learning models. We use it as a guide for our enhancements. It also helps us utilize diverse preprocessing methods in texts that we enhanced.

2.2.3.3. CICIDS2017 Dataset Description

The CICIDS2017 dataset is a good dataset developed by the Canadian Institute for Cybersecurity. It represents real-world network traffic. This file was created using CICFlowMeter, which uses PCAP files to extract flow-based features. CICFlowMeter is a tool used for generating the traffic flows which are developed for the CIC. It is used for developing 84 types of traffic features. It creates a graphical report after analysing pcap file. The data set consists of labeled network flows representing both benign and malicious activity. Researchers have developed and validated network intrusion detection systems

(IDS) using this data set.

The dataset has been taken from MachineLearningCVS (MLCVS) version as it contains CSV files that are labeled for use directly on the MachineLearning Models. It has provided details at flow level. This dataset version has 79 features that deal with many pieces of information about that packet length various statistics, protocol, and other flags relevant to the flow on the network.

2.2.3.3.1. Features and Label Information

The MLCVE Dataset consists of **84 features** which are a mixture of basic flow parameters and statistics along with network parameters. Such features can help classify normal versus malicious behaviors on the network. **Table 2.5** and **Table 2.6** below provides an overview of the features included in the dataset, along with their corresponding names in CICFlowMeter and a brief explanation. Feature Category is based on CICFlowMeter.

Table 2-5 Explanation of Features in the CICIDS2017 Dataset (Original Dataset)

Feature Category	Feature Name (CICIDS2017)	Feature Name (CICFlowMeter)	Description
Basic Flow Features	Flow ID	Flow ID	Unique identifier for a flow.
Basic Flow Features	Source IP	Source IP	IP address of the source.
Basic Flow Features	Source Port	Source Port	Port number used by the source.
Basic Flow Features	Destination IP	Dst IP	IP address of the destination.
.	.	.	.
.	.	.	.
.	.	.	.
Flow-based Timing Features	Idle Min	Idle Min	Minimum time a flow was idle.
Label	Label	Label	The class label indicating whether the flow is benign or belongs to a particular attack category

Detailed Table of Table 2.5 is at Appendix.

Table 2-6 Overview of Features in the CICIDS2017 Dataset (Original Dataset)

SNo	Feature Name (CICIDS2017)	SNo	Feature Name (CICIDS2017)	SNo	Feature Name (CICIDS2017)	SNo	Feature Name (CICIDS2017)
1	Flow ID	22	Flow Packets/s	43	Fwd PSH Flags	64	Down/Up Ratio
2	Source IP	23	Flow IAT Mean	44	Bwd PSH Flags	65	Average Packet Size
3	Source Port	24	Flow IAT Std	45	Fwd URG Flags	66	Avg Fwd Segment Size
4	Destination IP	25	Flow IAT Max	46	Bwd URG Flags	67	Avg Bwd Segment Size
5	Destination Port	26	Flow IAT Min	47	Fwd Header Length	68	Fwd Avg Bytes/Bulk
6	Protocol	27	Fwd IAT Total	48	Bwd Header Length	69	Fwd Avg Packets/Bulk
7	Timestamp	28	Fwd IAT Mean	49	Fwd Packets/s	70	Fwd Avg Bulk Rate
8	Flow Duration	29	Fwd IAT Std	50	Bwd Packets/s	71	Bwd Avg Bytes/Bulk
9	Total Fwd Packets	30	Fwd IAT Max	51	Min Packet Length	72	Bwd Avg Packets/Bulk
10	Total Backward Packets	31	Fwd IAT Min	52	Max Packet Length	73	Bwd Avg Bulk Rate
11	Total Length of Fwd Packets	32	Bwd IAT Total	53	Packet Length Mean	74	Subflow Fwd Packets
12	Total Length of Bwd Packets	33	Bwd IAT Mean	54	Packet Length Std	75	Subflow Fwd Bytes
13	Fwd Packet Length Max	34	Bwd IAT Std	55	Packet Length Variance	76	Subflow Bwd Packets
14	Fwd Packet Length Min	35	Bwd IAT Max	56	FIN Flag Count	77	Subflow Bwd Bytes
15	Fwd Packet Length Mean	36	Bwd IAT Min	57	SYN Flag Count	78	Init_Win_bytes_forward
16	Fwd Packet Length Std	37	Fwd PSH Flags	58	RST Flag Count	79	Init_Win_bytes_backward
17	Bwd Packet Length Max	38	Bwd PSH Flags	59	PSH Flag Count	80	Fwd Act Data Pkts
18	Bwd Packet Length Min	39	Fwd URG Flags	60	ACK Flag Count	81	Min Segment Size Forward
19	Bwd Packet Length Mean	40	Bwd URG Flags	61	URG Flag Count	82	Active Mean
20	Bwd Packet Length Std	41	Fwd Header Length	62	CWR Flag Count	83	Active Std
21	Flow Bytes/s	42	Bwd Header Length	63	ECE Flag Count	84	Active Max

2.2.3.3.2. Label Count Distribution

A count of various class labels was involved in the CICIDS2017 Dataset to get a further insight into their distribution. The count of this label indicates the imbalance between normal and malicious activities, which is vital while training ML models. **Table 2.7** provides a summary of both normal and attack records present in the dataset:

Table 2-7 Distribution of Labels in the CICIDS2017 Dataset

Traffic Type	Record Count
BENIGN	2,273,097
DoS Hulk	231,073
PortScan	158,930
DDoS	128,027
DoS GoldenEye	10,293
FTP-Patator	7,938
SSH-Patator	5,897
DoS Slowloris	5,796
DoS Slowhttptest	5,499
Bot	1,966
Web Attack - Brute Force	1,507
Web Attack - XSS	652
Infiltration	36
Web Attack - SQL Injection	21
Heartbleed	11

This distribution at **Table 2.7** reveals that the dataset is highly imbalanced, with certain attack types having significantly fewer samples compared to the benign traffic. Such imbalance could bias the model towards the majority class, making it less effective at detecting the other attack types. Specific preprocessing techniques, such as removing lesser data or resampling them, may be considered to handle this issue in subsequent steps. Still, resampling labels with the very low data such as Heartbleed would result on producing

synthetic data. It would also have a high probability of overfitting because of lack of the data to resample.

2.2.3.3.3. Feature Dropping

We drop features of the original data for training to make the model more efficient and generalizable. We decided to take off some features because they don't help predict anything, they make the model feel too similar to the example we train, or they just bias the model to memorize and not generalize.

To start off, features that had constant values throughout the dataset were dropped. These constant features are **Fwd URG Flags**, **Bwd URG Flags**, and **URG Flag Count**. Since these features did not vary across different network flows, they offered no useful information for distinguishing between benign and malicious traffic. Including such redundant information could potentially increase computational overhead without providing any added value in terms of model accuracy. By removing these features, we aimed to reduce unnecessary complexity and ensure that the machine learning algorithms focused only on features with meaningful variability.

Next, certain unique identifiers were also dropped, such as **Flow ID**. The **Flow ID** uniquely identifies each flow but has no underlying pattern that could generalize across different flows. Since the goal of machine learning is to identify patterns applicable to unseen data, retaining features like **Flow ID** would likely cause the model to overfit. The model would end up "memorizing" specific flow identifiers rather than learning generalizable features that distinguish between benign and malicious activity.

Also, the features that might have worked as shortcuts for the model was removed. The Source IP (Src IP), Destination IP (Dst IP), timestamp, etc. were also part of it. These attributes may allow the model to pick up on unintended shortcuts that do not relate to flow behavior, but have to do with malicious activities' addresses or dates. For example, this model could learn to associate IP addresses with attacks, reducing its effectiveness if IPs seen in the real world are new. In the same way, Timestamp might make the model link the timing of attacks and could help in the identification of attacks any only at the time of a certain campaign. Such behavior may limit the model's detection of unfamiliar attacks and impact generalizability. By taking away these attributes, the model would learn better based

on malicious activity patterns rather than meta-data specifics which made it robust.

Lastly, the Source Port (Src Port) feature was removed due to its randomness. Ports are assigned dynamically, so using Source Port to train the model could lead to overfitting. That is, specific numbers will be associated with attack and benign flows, and that is without any valid behavioral reason. When met with different port numbers, they will falsely detect such packets as attacks. This risk was mitigated by dropping the Source Port, to prevent the model from making false correlations.

During the feature dropping, we also check the shape of the dataset. At first, if from 84 features 6 features are removed, then 78 features will be in the final dataset. When I checked the shape of the data frame my dataset had more rows/columns. I expected to have 78 features, but I ended up having 79 instead. So, I checked the features list of the original dataset. I found a column called Fwd Header Length.1 This column was the same as 'Fwd Header Length'. Hence, 'Fwd Header Length.1' was deleted eliminating the mismatch. The dataset was finalized and made ready for preprocessing after this minor correction.

Omitting these features was a must to ensure that the models trained on the data have the best chance of generalising well beyond the training data. This ensures that they detect novel attacks rather than just overfitting to values, metadata, or situations. To summarize, this preprocessing was needed to move forward with the model development to achieve a complexity/accuracy/robustness balance and for improving its reliability.

The CICIDS2017 Dataset developed in this research will be the baseline dataset for proposing and evaluating the intrusion detection models. This system excluded features which could introduce redundancy or overfitting or present obstacles to generalization, while retaining key flow-level features extracted from CICFlowMeter. This version of the dataset was a baseline, being the closest to the original in data but with certain omissions to improve model quality. One by one, preprocessing techniques were applied to the dataset.

2.2.4. Dataset Preprocessing for Clean Dataset Creation

This part focuses on creating a clean dataset for model training purposes.

2.2.4.1.Overview of Preprocessing Goals

The goal of this cleaning process was to create a clean and balanced dataset for training the model effectively. Though the CICIDS2017 dataset was integrated and prepared, we still had issues like imbalanced labels, underrepresented classes and data inconsistency. This prep process is designed to cut down on these problems through label cutback, removal of repeated or erroneous data, and similar fixes on the missing or invalid data so that the final dataset produced a reliable one.

2.2.4.2.Label Reduction and Data Imbalance Mitigation

Major problem with the CICIDS2017 dataset was the extreme imbalance in the distribution of labels. According to the earlier Label Count Distribution (Table 2.7), some attacks had far lesser samples than other attacks and would have rendered a very imbalanced dataset if employed as is for model training. In particular, the attack types Bot, Web Attack – Brute Force, Web Attack – XSS, Infiltration, Web Attack – SQL Injection, and Heartbleed accounted for only 4,193 records. The underrepresented classes formed minor proportions of the entire dataset relative to well-represented classes like BENIGN or DoS Hulk.

Having so few records for these labels may have caused a lot of bias in the model. If a model is trained with labels that are infrequently seen, it will miss out on or incorrectly label these types of attacks, making it unable to detect actual network attacks. Therefore, it was decided to eliminate these branches from the dataset and use only those branches which were adequately represented. By using this label reduction process, they were able to remove data sparsity, simplify training and improve generalizability.

2.2.4.3.Data Redundancy Check and Reduction of BENIGN Records

This part focuses on data redundancy and reduction of BENIGN records for a more balanced dataset.

2.2.4.3.1. Data Redundancy Check

First thing I did after minimizing the under-represented labels was to check for redundant or duplicate instances in the dataset. In case they are present in the dataset, they will unwantedly bias the model by reinforcing certain representations. Comprehensive data redundancy check with duplicate rows will be performed for this purpose. **Table 2.8** records the label count of duplicate rows found in the original dataset.

Table 2-8 Label Count for Duplicate Rows in the Original Dataset

Traffic Type	Duplicate Count
BENIGN	236,257
PortScan	101,501
DoS Hulk	59,564
SSH-Patator	2,826
FTP-Patator	2,457
DoS Slowloris	507
DoS Slowhttptest	323
Web Attack - Brute Force	62
DDoS	20
Bot	19
DoS GoldenEye	14

The data redundancy check found many duplicates in the data set for several labels. The label BENIGN had 236,257 duplicate records alone in the dataset. This is a huge part of the dataset and would have caused biased learning. PortScan and DoS Hulk showed high redundancy with 101,501 and 59,564 repeated records respectively. These duplicates were removed so the dataset would represent true, diverse network traffic. The model worked better on new data because it was trained with data of better quality, as redundant entries have been removed.

2.2.4.3.2. Reduction of BENIGN Records

After balancing the data, the next process targeted the huge presence of BENIGN records in the data. Over 2.2 million records were provided in the BENIGN label which comprised of an extremely large number of records. This imbalance could cause the m i t rs to learn too much towards BENIGN classification making it difficult to learn about malicious activities.

To deal with this, BENIGN records were reduced randomly such that they can be balanced across different classes and not biased. By utilizing this technique, a great deal of BENIGN instances were retained so that the model is able to learn about its behaviour while also preventing this label from dominating the learning. The overall model accuracy for all traffic types (benign and malicious) was enhanced by decreasing the number of BENIGN samples. In addition, with proper class balance, the model would be able to learn better and identify more patterns of attacks without being biased towards the majority class.

2.2.4.4.Handling Missing and Invalid Data

Once the labels got reduced and ensure lesser data redundancy, the next step was to take care of inconsistency and invalid values that can degrade model training. We cleaned the data during this step to remove any entries that may compromise the robustness of the data set.

Rows with NaN values were deleted. NaN denotes missing or undefined information. Thus, NaNs in a dataset would yield incomplete or inaccurate training information and thus prediction. Dealing with NaN values was crucial for having a complete sample for training purposes.

Rows with Infinity (Inf) values were also discovered and deleted in addition to NaN values. The value is often due to numerical operations, like a zero divided by zero or overflow errors. Keeping Inf numbers will make the model unstable, as ML algorithms do not understand the Inf value. It was important to eliminate them which ensured numerical stability during training.

They also removed negative values from the data. Negative values are logically impossible in many of the features represented in the CICIDS2017 dataset like packet sizes or flow durations. The negative sign indicates some mistake either in data collection or data

corruption and cannot train the model as it is not the logical or accurate answer. Therefore, these were removed for the integrity of data.

Finally, any empty row in a dataset has also deleted. These rows give no useful information and will only provide noise, adding unnecessary computation during training.

Based on the steps of data cleaning together the final set of data was valid and consistent. The cleaned dataset was used for building a machine learning model that can detect all types of intrusions in the network.

2.2.4.5.Splitting the Cleaned Dataset

After the dataset has been cleaned and processed thoroughly, the last step was to the train-test split of the dataset. The motive behind the split was to have the ability to create and evaluate machine learning models in an unbiased way. The model was able to work well on new data that it wasn't taught on, meaning it could do its job in a real-world application.

Training Dataset which was utilized to train the ML models and it helped the model learn the patterns which are present in the network traffic or classifying the traffic as BENIGN or malicious. In addition, a different Test Dataset was kept to examine model efficiency which estimates how well the model would generalize to data it had never seen before.

The cleaned dataset was split in such a way that the classes were balanced across the training and test datasets so that both subsets trained on all labels fairly. Table 2.9 shows the label distribution among the training and test datasets:

Table 2-9 Label Count in the Training and Test Dataset After Cleaning and Splitting

Label Name	Train Dataset Record Count	Test Dataset Record Count
BENIGN	120,362	30,174
DDoS	65,180	16,296
DoS GoldenEye	6,167	1,542
DoS Hulk	130,810	32,771

Label Name	Train Dataset Record Count	Test Dataset Record Count
DoS Slowhttptest	1,677	465
DoS Slowloris	3,056	831
FTP-Patator	4,930	1,288
PortScan	92,109	31,748
SSH-Patator	2,576	1,176

These tables describe the balance obtained among various classes in training and test datasets. The BENIGN label still had enough records, but it was reduced to a significant amount compared to other labels so that it doesn't dominate the learning. This balance in the representation means that the model has sufficient samples from both these classes so that it may learn to distinguish between not just the benign traffic but also the many types of attacks. The data pre-processing steps explained in this section were a key factor for converting the raw CICIDS2017 dataset into a reliable representative dataset that could be used to train an ML model.

As a result of these endeavours, the final dataset was created to be clean, consistent and adequately balanced so as to be a good base for building a strong model of intrusion detection. The machine learning model was expected to have enhanced accuracy, generalizability, and reliability for the detection of common and rare network intrusions by solving data quality issues in totality.

2.2.5. Preprocessing Dataset with Destination Port Feature Removal

This part focuses on preprocessing dataset towards Destination Port feature removal.

2.2.5.1.Rationale for Further Preprocessing

The next phase of data preparation focused on intricate data operations where cleaning and splitting already executed in Section 4 were executed with only one modification. The Destination Port feature was removed from the dataset. They took this decision to avoid

manipulation in the Destination Port as it was found easy to manipulate this port making a no good feature to distinguish benign and mediocre traffic in ML.

Attackers can easily manipulate both Source Port and Destination Port. Before Source Port got removed for the same reason. Then, the model may become overfit. These are correlations valid under one configuration of the network but not under all configurations over the environment. By taking away the Source Port and Destination Port, it forced the model to learn the invariant and meaningful behavior of the network traffic instead of relying on association based on port information that may be misleading.

2.2.5.2.Increased Data Redundancy After Destination Port Removal

After disabling the Destination Port feature, it could be seen that the original dataset had a lot of redundant data, especially the BENIGN label. In the absence of Destination Port, many network flows that previously had small differences became identical, thus increasing the number of duplicate rows. In particular, the number of duplicates rose to 687424 from 404564 during the last cycle of Preprocessing. The duplicate row frequency after the Destination Port has been dropped is given in **Table 2.10** :

Table 2-10 Label Count for Duplicate Rows After Dropping Destination Port

Traffic Type	Duplicate Count
BENIGN	464,385
PortScan	157,328
DoS Hulk	59,564
SSH-Patator	2,826
FTP-Patator	2,457
DoS Slowloris	507
DoS Slowhttptest	323
DDoS	20
DoS GoldenEye	14

The BENIGN masking tag had a lot of redundancy inserted with 464,385 duplicates, almost double the original. In the same way, the PortScan label highly increased redundancy, with

157,328 records found to be repeated. The increased redundancy warranted extra processing to check for duplicates that might add slackness to the entire process.

2.2.5.3.Handling Increased Redundancy and Enhancements Over the Previous Stage

Duplicate entries were also removed in the same way as we did earlier in view of the increased data redundancy. To ensure that the data remains free from duplication that could alter model treatment, 687,424 identified duplicates were dropped from the data.

Apart from duplicates removal, the overall preprocessing strategy was kept the same as in Section 4. That is, we reapplied label reduction to remove underrepresented classes and randomly reduced the BENIGN label to maintain the balance thereafter. We cleaned the remaining inconsistencies like any NaN value, Inf value, negative value and empty rows.

The big improvement over the last stage was designed to help the model generalize better. To ensure that the model did not learn to rely on the Destination Port, this feature was removed to prevent easy manipulation. eliminating both Source Port and Destination Port meant that we would have to focus on complex features that indicated malware, as opposed to simple features where a port is correlated with a traffic type. Its objective was to ensure that the model learnt robust features that will generalize to different network conditions.

The extra discontinuity indicated that Destination Port helped to differentiate flows that were otherwise similar. By eliminating this feature and handling the resultant duplicates, the shuffled dataset was made to be more uniform, in terms of flowing behaviour and traffic characterization rather than using metadata like port numbers.

2.2.5.4.Splitting the Cleaned Dataset After Destination Port Removal

After managing the increased redundancy and finalizing data cleaning, the dataset was split into training set and test set. The subsets used for developing machine learning models and the subsets used for testing will be distinct and split 80:20.

Both datasets had a balanced class which means all the labels were present so the learning would take place effectively on a wide range of network behaviours. The label distributions of the training and test datasets after the removal of the Destination Port and further cleaning are presented in **Table 2.11**:

Table 2-11 Label Count in the Training and Test Dataset After Removing Destination Port and Cleaning

Label Name	Training Dataset Record Count	Test Dataset Record Count
BENIGN	115,400	30,174
DDoS	65,180	16,296
DoS GoldenEye	6,167	1,542
DoS Hulk	130,810	32,771
DoS Slowhttptest	1,677	465
DoS slowloris	3,056	831
FTP-Patator	4,930	1,288
PortScan	1,382	31,748
SSH-Patator	2,576	1,176

The tables show that even though removing Destination Port results in an increase in redundancy, the cleaned and balanced training and test sets have a similar distribution to the preceding case. Keeping this balance is essential to allow the model to properly learn the properties of both benign and malicious traffic without being biased towards any particular class.

Elimination of Source Port and Destination Port feature is an important enhancement in the preprocessing phase. These features are easy to manipulate, which may cause the model to form incorrect associations.

This may reduce the model's effectiveness in the real world. The model was better equipped to detect behavioral patterns and traffic characteristics because it was not fed port-related information at all; that info could easily be spoofed by an attacker. It was expected that this method would result in a more generalized and stronger intrusion detection model that could detect various kinds of intrusion on the network under different network situations.

2.2.5.5.Resulting Datasets of MLCVE_clean and MLCVE_clean_dest

As a result of the preprocessing steps undertaken in **Section 4** and **Section 5**, two distinct datasets were created:

1. **MLCVE_clean**: After the pre-processing described in Section 4, the label reduction, duplicate rows, random downsampling of the BENIGN class, general cleaning of the

data to get rid of inconsistencies, was done to get this dataset.

2. **MLCVE_clean_dest**: This dataset incorporates all the preprocessing steps performed on **MLCVE_clean**, with the key distinction that the **Destination Port** feature was also removed. The rationale for dropping **Destination Port** was to reduce the risk of overfitting due to easily manipulated metadata and to ensure the model to learn more generalized patterns of network traffic behavior that are less likely to be tied to specific ports.

These two datasets MLCVE_clean and MLCVE_clean_dest aim to analyse how removing the Destination Port affects the generalizability and robustness of the model. By checking how well the machine learning models work on the two datasets, we can see how significant Destination Port is for a network intrusion detection task. We can also see whether removal of Destination Port can help create a robust generalizable model. The findings of this comparative study will help decide if models should learn deeper flow based behaviour rather than metadata which can be easily modified.

2.2.5.6.Creation of a Binary Classification Dataset

To simplify the classification task further and to assess the effectiveness of anomaly detection regardless of the attack type, a binary classification dataset was created from the two previously processed datasets MLCVE_clean and MLCVE_clean_dest. According to this strategy, all traffic will be labelled either as normal (benign) or abnormal (anomalous) instead of classifying between the types of attacks present in the dataset. The dataset is suitable for models that only have to differentiate between ‘good’ traffic and ‘bad’ traffic.

In the original datasets, the labels signified the type of attack. Hence, a multi-class classification. Nonetheless, the labels in the binary classification dataset were combined to simplify classification. The label BENIGN was unchanged and assigned 0, meaning normal traffic. The other labels representing different attack types were put under one label called ABNORMAL which will be assigned as 1. This includes other attacks such as DDoS, PortScan, DoS Hulk, FTP-Patator etc. which were labelled as abnormal.

This made the classification task easier since we would have to tell if the given traffic instance was normal or malicious instead of many classes of attacks.

2.2.5.6.1. Label Distribution in the Binary Dataset

Following the conversion, the **MLCVE_clean_dest** dataset was split into **training** and **testing** datasets. Below **Table 2.12** is the label distributions for both splits:

Table 2-12 Label Counts in the Binary Training Dataset:

Label	Training Dataset Count	Test Dataset Count
BENIGN (0)	115,400	30,174
ABNORMAL (1)	215,778	86,117

The binary training data set consist of 215,778 instances of Abnormal and 115,400 instances of Benign. Thus, the dataset still remained a bit unbalanced, that is abnormal traffic was more than normal traffic. Likewise, there were 86,117 ABNORMAL instances and 30,174 BENIGN instances in testing data.

A binary classification dataset was created using MLCVE_clean and MLCVE_clean_dest which simplified the intrusion detection problem. Transforming the dataset into a binary format means the focus shifted from distinguishing one attack from the other to classifying whether the traffic is benign or malicious. This shift seems appropriate for various usages where the focus is on spotting any dubious activity without the need for attack type specification. The resulting dataset is still imbalanced but presents a more balanced learning opportunity compared to the original multi-class datasets, thus allowing the training of binary classifiers for generalized anomaly detection.

2.2.6. Model Training

Model training will be explained in here.

2.2.6.1.Training the Dense Neural Network (DNN)

To build a strong intrusion detection model TensorFlow Keras was used to implement a Dense Neural Network (DNN) The training process is flexible or can be changed depending on the outcome of the feature selection and feature importance analysis for obtaining better efficiency, generalizability, and performance of the model.

2.2.6.1.1. Data Preparation and Feature Selection

The training datasets MLCVE_clean and MLCVE_clean_dest were preprocessed as described in previous sections. An optional feature selection technique based on the importance score was included in the pipeline of training. To facilitate dynamic optimization, properties that were not integral to the construction of the empirical model were not used in training.

We get score importance through a permutation-based approach on previously trained model. The features were ranked according to their score and utilized to generate a prediction. The feature selection mechanism was designed for on/off functioning, depending on the specific experimental demands. For this training, we set the feature selection to false, which means we will not drop any features and will use all. This gives the model the most extensive input features, providing it with more information present in the network data used in probing.

But using feature selection, the architecture of the model could also change the layers. The most important features were prioritized, which might have lowered the input dimensionality, thus causing fewer neurons in the input layer. This strategy made sure that the model concentrated on the most important features, which lessened the risk of overfitting and made the training more efficient.

An example of feature importance computed from a trained model using **MLCVE_clean** is shown in the following **Table 2.13** here:

Table 2-13 Example of Feature Importance Scores

Weight	Feature
0.2121 ± 0.0003	Packet Length Mean
0.1951 ± 0.0002	PSH Flag Count
0.1829 ± 0.0002	Average Packet Size
0.1076 ± 0.0003	Packet Length Std
0.0860 ± 0.0001	ACK Flag Count
0.0688 ± 0.0002	Fwd IAT Total

Weight	Feature
0.0672 ± 0.0002	Destination Port
0.0613 ± 0.0001	Bwd IAT Std
0.0493 ± 0.0000	Bwd Packet Length Std
0.0484 ± 0.0001	Fwd IAT Std
...	...

The Destination Port is a significant feature appearing in the table with good feature importance of 0.0672 ± 0.0002. Although the Destination Port might be manipulated, it still has useful information to distinguish between various network flows. The Destination Port serves dual purposes. Its operational nature can be helpful for detection, however, its failure to detect was the cause for the removal phase of preprocessing for MLCVE_clean_dest. In this research, the trade-off between utility and robustness was a key consideration that drove different preprocessing paths to investigate the impact on model performance.

The feature importance scores showed that features like Packet Length Mean, PSH Flag Count, Average Packet Size, Destination Port were significant predictors in the model, while the other features were not that significant. It was possible to use this kind of information to reduce the input layer size and become a streamlined model focusing on the most essential ones.

2.2.6.1.2. Dense Neural Network Architecture

The architecture of the Dense Neural Network (DNN) was designed to adapt based on the selected number of features. Designed architecture consisted of:

- **Input Layer**: The input layer size is based on the number of features that the feature selection step retained. If feature selection is off, we take all features (ALL). But when feature selection was performed, the input layer was automatically set to be compatible with the reduced dimensionality of the selected features.
- **Hidden Layers**: The used model has three hidden layers of 128, 64, and 32 neurons. Every one of those layers used a ReLU (rectified linear unit) activation function that reduces the vanishing gradient problem. In addition, it helps speed up the training.

- **Output Layer**: The output layer had 9 neurons representing different classes of network traffic (benign and malicious). We use the softmax activation function which generates a probability distribution over all 9 classes for multi-class classification.

The **Adam optimizer** was employed for model compilation. Its adaptive learning rate capability was particularly useful in handling the sparse and noisy nature of network data. The **loss function** used was sparse categorical cross-entropy, well-suited for the multi-class classification problem.

2.2.6.1.3. Model Training Process

The training of a model follows certain steps and considerations to enhance the model performance for it to be robust. Initially, dynamic feature selection contributed greatly to the DNN input layer configuration. According to the feature importance scores, input dimensionality was optimized. By performing feature selection, only the most significant features were kept to make the model less complex, and less computationally intensive. For instance, if only 20 of the original 84 features are found to be significant, the size of the input layer is modified. The model became simpler and the model training was focused on important features eliminating noise and hence limiting overfitted data.

The dataset gets used for training and validation set. The model parameters were fitted using the training set, while the validation set was used as an independent set to evaluate the model's generalization ability during training. By checking the model's output on the validation set, it was easy to tune the hyperparameters in a dynamic way to avoid overfitting on the training set.

To train the model effectively, a batch size of 256 was used and training was done for 250 epochs. Due to the large batch size being used, the gradient estimates were more stable. Hence, the optimization was easy. Having an access to around 250 epochs for training is something that helps the model to get converged. To prevent overfitting, we used early stopping methods in our model. These systems allowed the training to stop once the model had shown satisfactory performance. Thus preventing the training from continuing so far that it would start to memorize.

The unequal number of data classes created a problem since BENIGN class was substantially larger than the attack classes. Class weights were assigned to the model to ensure enough importance is given to the minority classes. More weight was given to the underrepresented labels DDoS or SSH-Patator, allowing better learning of the characteristics of these classes. The main reason behind this was to prevent the model from being biased towards the majority.

The training also makes sure that the data is shuffled in every epoch. So it shuffled the data to ensure that the model does not learn any unintended sequence-based correlations in the data. It was especially important as network traffic data can be inherently sequentially or temporally dependent. The training samples were shuffled for each epoch so that the model sees a different sequence of examples every time.

The training process was further improved by two callbacks: early stop on F1 score and learning rate reduction on plateau. The early stopping callback was set up to keep track of the model's validation set F1 score. The F1 score, which encompasses precision and recall, result in a balanced metric to stop training when the model was at a desirable level across all classes. F1 score balance was set as macro, resulting in same importance to each class result instead of true instances from the data-set. This allowed the model to keep the F1 score optimized for better generalization instead of letting the model focus on labels with high numbers of instances within the dataset. We set the learning rate ourselves and it would halved ever 5 steps when if there are no improvements in F1 score. The model's learning score was set to a low value, as well, so as to reduce chances of getting stuck in local minima. This was particularly useful for an unbalanced data set which would give false results if optimized only for accuracy. The learning rate reduction callback slowed down the learning rate whenever validation loss plateaued. When a set of epochs did not yield an improvement, early stopping would terminate the training and reset the weights to the best of the f1 scores from the training run. This reduction in dynamic allowed the model make more finer adjustment during the liminal stages of training to make convergence more smooth.

Training of the Dense Neural Network was performed using a combination of dynamic feature selection, class weighting, early stopping, learning rate modifications, and shuffling of the data. All of these measures combined ensured that the model was still able to learn from sufficient and diverse examples without requiring much computational power. With

the dynamic tuning of the input layer according to feature importance, class weighting, and adaptive training callbacks, the resulting model is strong and capable of detecting a common and rare network threat effectively.

2.2.6.2.Training the Random Forest Classifier

In addition to Dense Neural Network another model called Random Forest Classifier was used for Intrusion Detection. For tabular data, Random Forest algorithm is very suitable. This gave a good way to model the complex patterns from the network traffic data. The goal was to evaluate its performance against the neural net based method for acknowledging the best model for intrusion detection.

2.2.6.2.1. Data Preparation and Feature Selection

The MLCVE_clean datasets and MLCVE_clean_dest were used to train the Random Forest classifier as discussed in previous sections. Feature selection was useful in random forest model to enhance the performance. Random Forest has feature importance metrics, which allowed us to rank features based on their contributions to the decision-making process of the model.

A Random Forest model was trained on the traffic data and the importance values of the features were derived. This was done to check which features were more important when distinguishing between malicious and benign traffic. The following **Table 2.14** ranked features emerged from analysis:

Table 2-14 Feature Importance Scores from Random Forest Model

Rank	Feature Name	Importance Value
1	Destination Port	0.0680
2	Init_Win_bytes_backward	0.0624
3	Fwd Packet Length Max	0.0349
4	Flow IAT Mean	0.0345
5	Init_Win_bytes_forward	0.0314
...	...	...

Destination port feature was most important feature with importance value of 0.0680. This characteristic shows impressive power to separate the different network activities, and it also had a considerable feature importance during the training of the neural network. Destination Port was removed from the MLCVE_clean_dest dataset because it is not robust enough as it can be manipulated. It ranks quite highly here which suggests that it may be useful for classification in some cases. Thus, it can be anticipated that the Destination Port drop in the MLCVE_clean_dest dataset will decrease the Random Forest model performance. If this feature is not present, then the model may not be able to classify some network flows accurately.

Init_Win_bytes_backward, Fwd Packet Length Max and Flow IAT Mean are some other features that scored high which relevant in identifying different types of networks traffics. At the feature selection step, it was possible to reduce the number of features before actual modelling took place. Thus, certain features were chosen to be left out because they had little effect on the classification. This method made training easier by reducing computation as well as noise by less important features which helped to give more robustness to the model.

2.2.6.2.2. Random Forest Model Setup

The Random Forest Classifier employed in this study was implemented using the scikit-learn library and aimed at capturing non-linear relationships in network traffic via ensemble learning. A Random Forest is comprised of decision trees that independently classify data and vote on the overall output. This kind of model is resilient and works well for datasets that have a lot of dimensions.

The model was set up with different parameters to make it fairly accurate and doable. We set the number of decision trees (estimators) to 100. This balance of prediction quality and training performance worked well. I chose Gini impurity as a criterion for splitting the nodes, which helps in finding the best possible splits. Each tree is allowed to grow unrestricted in max depth in order to capture all the details of the data as possible. It is still worthwhile to try different max depths in order to better generalize the model. The model was also trained using max depth for better generalization and to compare the results.

Random forests aren't liable to overfitting as the base decision tree. It gives a final decision based on several trees. Through averaging, variance is reduced while generalizing

capabilities improve as well.

2.2.6.2.3. Model Training Process

The Random Forest model training involves the splitting of training dataset into training and testing datasets. The Random Forest model was fitted on the training set and validated on the independent test set for Generalization evaluation.

A big plus of Random Forests is that no scaling is needed and they only require unbiased features. Many models like neural networks require feature scaling. So your end result of your preprocessing will go directly to training in very common ML and DL libraries without further normalization and standardization.

To cope with class imbalance, class weights were assigned dynamically based on label distribution in the dataset. In particular, the class_weight parameter was set to 'balanced', which 'balances' the weights inversely proportional to the class frequencies in the training data. This ensured that the minority class like DDoS, DoS Slowloris and others were given enough importance during model training and not overshadowed by the majority class which is BENIGN. This was necessary to balance the model that was capable of detecting both frequent and rare types of attacks.

While training, another robustness was added to the Random Forest model through the use of a random subset of features for the construction of each decision tree. Random forest avoids overfitting by using random feature selection for each tree, which results in a more generalizable model. Randomized feature selection results in less chance of overfitting to a particular set of features. Moreover, limiting each node splitting to a certain max number of features made sure that the model was not very complex so as to learn noise by heart but rather learn the patterns.

2.2.6.2.4. Feature Importance and Optimization

The random forest model was optimized through feature importance analysis. After training the model, the feature importance scores were extracted to obtain insights regarding the most important features for classifying. According to Table 6.2, the Destination Port, Init_Win_bytes_backward, Fwd Packet Length Max, and Flow IAT Mean were the most influential features.

Destination Port has a high ranking, suggesting that it can help differentiate activities associated with different networks, but it may be susceptible to attacker manipulation. This insight shows that it is important to select the right features for training the model. Manipulatable features can also provide significant predictions under certain circumstances.

Analysis of Features Importance also helped in retraining the model which could exclude the optional features with lower score in further model training. A retrain of the Random Forest classifier with a reduced number of features will make it computationally less intensive. That is, the training time will be reduced with a high level of accuracy. The model became more robust and safer from being overfitted as a result of this streamlined approach to make the model focus more on it.

The training of Random Forest Classifier was done in an orderly manner using feature importance method. The model will use dynamic feature selection, class weighting, and random feature sampling to learn from the train data but not overfit.

it can be seen that since Destination Port is an important feature in Random Forest the removal of this feature in MLCVE_clean_dest will affect the performance of the model negatively If this functionality is not available, the Random Forest will not work as accurately specifically with respect to some attack instances and benign traffic relying on port-based functionality. The final aim of using Random Forest was to obtain a classifier that is robust yet versatile whose main aim is to detect and classify any sort of network attack.

2.2.6.3.Training on the Binary Classification Dataset

After training the MLCVE_train dataset with multi-class classification models, the dense neural network (DNN) model and the random forest classifier were also trained on MLCVE_clean and MLCVE_clean_dest binary classification dataset. The binary dataset merges all attack types into the same label ABNORMAL and keeps BENIGN as the label for normal traffic. This change made classification easy to evaluate general anomaly detection capabilities.

2.2.6.3.1. Application of Multi-Class Model Training Configurations

The dense neural network and random forest models were trained on the binary dataset using the same configurations that had been optimized for the multi-class classification case. The

aim was to use the best configurations to find out how well these models perform in a binary classification, that is, BENIGN and ABNORMAL network traffic, or rather, good and bad traffic.

The Dense Neural Network had the same architecture and train-ing process with just a change to the output layer which was made suitable for the binary classification task. The output layer was modified to have a single neuron with a sigmoid activation function which allows the normal or anomalous class of a traffic flow. The hidden layers, batch size, epochs, early stopping and class weighting settings were kept the same as found in the original optimized version to keep the new and previous processes consistent.

Likewise, the parameters that secured the best outcome during multi-class training were utilized to train the Random Forest Classifier. The number of estimators, criterion for node splitting, and class weighting all remained the same that is identical to the earlier configurations considered optimal. The Random Forest used the ensemble learning method to combine outputs of individual decision trees to predict the output of BENIGN as ABNORMAL.

The binary classification approach narrowed the problem down to abnormal behavior detection only, regardless of what type of attack it is. It emulated real-world use cases where intrusion detection systems mainly focused on flagging suspicious activity. By using the same optimized parameters in the binary dataset, the generalization capacity of both DNN and Random Forest models was assessed with regard to their ability to recognize any form of network anomaly.

2.2.7. Model Architecture Updates

In this section, tests were done in order to observe the effect of different architecture design and configurations for DNN model.

2.2.7.1.Layer Configuration Updates

Following Dense Neural Network layer configurations at **Table 2.15** were tested as an alternative to existing architecture configuration: Bottleneck Architecture, Sparse Wide Architecture, Pyramid Architecture, Regularization with Batch Normalization, Gradual Compression with Regularization, Single Hidden Layer (32 neurons), Single Hidden Layer

(64 neurons), Single Hidden Layer (128 neurons). The table below gives a brief explanation of the tested layer configurations.

Table 2-15 Alternative Layer Configurations

Configuration	Full Name	Architecture	Features
BottleNeck	Bottleneck Architecture	128 → 32 → 64 → 9	Narrow bottleneck layer for compression, expansion for reconstruction
Sparse Wide	Sparse Wide Architecture	256 → 64 → 128 → 32 → 9	Alternating wide and sparse layers for balanced feature learning
Pyramid	Pyramid Architecture	512 → 256 → 128 → 64 → 32 → 9	Gradual reduction with wide initial layers for feature diversity
Reg. Batch Norm.	Regularization with Batch Normalization	128 → BatchNorm → 64 → BatchNorm → 32 → 9	Batch normalization for stable training, dropout for regularization
Gradual Compression Reg.	Gradual Compression with Regularization	256 → 128 → Dropout → 64 → 32 → Dropout → 9	Dropout regularization with gradual compression to reduce overfitting
Single 32	Single Hidden Layer (32 neurons)	32 → 9	Minimalistic single hidden layer with 32 neurons

2.2.7.2.Optimizer Configuration Update

The learning rate optimizer used in the architecture was the Adam optimizer. Following **Table 2.16** shows the alternative optimizers tested configuration update.

Table 2-16 Optimizer Configuration

Optimizer	Learning Rate	Additional Features
SGD	0.001	Momentum=0.9 (Accelerates convergence)
RMSprop	0.001	Handles noisy gradients, useful for RNNs
Adagrad	0.001	Adapts learning rate for each parameter, good for sparse data
Adadelta	1.0	Addresses Adagrad's aggressive decay problem
Nadam	0.001	Combines Adam and Nesterov momentum

2.2.8. Automated Real-Time CICFlowMeter Filtering IDS

The Automated Real-Time CICFlowMeter Filtering IDS is a system for real-time detection of network attack tools. The above goal was to design a scalable system that can capture, analyze and classify network traffic using trained models based on flow-based features from the CICIDS2017 dataset. The system underwent careful refinement over four distinct iterations. Each new version added one or more new detection functions to rectify the shortcomings of the previous one. The whole development and deployment of Automated Real-Time CICFlowMeter Filtering IDS took place on an NVIDIA Jetson AGX Orin. This platform was chosen due to its powerful computing capability for real-time edge processes.

2.2.8.1.Version 1: Foundational Real-Time IDS System

The first version of the system was able to introduce a model for an automated real-time intrusion detection which performed the flow-based extraction of features from traffic using CICFlowMeter and classified the output using Random Forest. At first, we used Tcpdump to capture network packets and generate Packet Capture (PCAP) files that were processed sequentially by CICFlowMeter to produce flow features. We decided to use Tcpdump to capture packets as we noticed the built-in capture mode of CICFlowMeter was prone to dropping packets.

The first version has main functions that include filter_and_rename_columns(df) and filter_and_rename_columns_reverse(df). They are meant to ensure that the features from CICflowmeter are consistent with the Random Forest model. Still, because these processes are carried out on a sequence, like capturing packets, extracting features, and predicting, a

large loss of packets was noticed. This happened because while the previous packets were being processed, there was a possibility of missing packets from the flows.

2.2.8.2.Version 2: Enhanced Concurrent Processing and Threading

Version 2 improved on Version 1's limitations with multi-threading that allows concurrent processing. Two threads are used to not lose the packets while processing. One thread is used to capture the packets and the other thread is used to extract features and prediction. This made sure that the capture could go on without interruption and looking at previous packets.

The start_sniffing() function was responsible for continuous packet capturing and the process_pcap_with_cicflowmeter() function monitored the directory for new PCAP files to perform automated feature extraction and prediction tasks. The use of concurrent processing was key to allowing the IDS to maintain a real-time character without affecting the accuracy or completeness of network flow analysis.

2.2.8.3.Version 3: Flow Management with Scapy

Version 3 went further and added scapy. Scapy is a Python library for sending and capturing packets. In other words, we improved our data stream efficiency. Version 3, unlike previous versions that concentrated on bulk PCAPs, gave more granular flow-level control. The function track_flow(flow) was implemented to monitor individual network flows to manage their life-time. None of the flows will last more than 100 seconds.

This version significantly reduced latency in the analysis. As soon as flows were processed when they were completed or reached the timeout. This change enabled the system to be much more dynamic by allowing it to focus on the completion of individual flows rather than processing all of them in bulk.

2.2.8.4.Version 3 Variant: DL Integration with TensorFlow (v3_2)

The **v3_2** variant aimed to leverage the power of deep learning by integrating a Dense Neural Network (DNN) using TensorFlow. Given the dependency issues between CICFlowMeter and TensorFlow, a dual-environment solution was adopted. The **cic_env** was used for packet capturing and feature extraction, while **tensor_env** was utilized for TensorFlow-based model prediction.

The **make_prediction_in_tensorflow_env()** function facilitated the execution of the TensorFlow model in a separate environment to avoid conflicts. Predictions were generated using **tensorflow_predict.py**, which performed feature scaling, model inference, and label decoding. The adoption of TensorFlow allowed for improved detection of complex, non-linear patterns in the data, enhancing the system's capability to identify sophisticated attacks.

2.2.8.5.System Implementation on NVIDIA Jetson AGX Orin

To ensure proper functionality, all versions of the Automated Real-Time CICFlowMeter Filtering IDS were deployed on the NVIDIA Jetson AGX Orin platform. We have selected the Jetson AGX Orin, which has powerful computational capabilities and with its GPU and CPU combination, it can handle real-time data capture, feature extraction, and model inference. This platform can perform many machine learning tasks at the edge, eliminating the delay that occurs when sending data to a central server for processing. It was able to use TensorFlow for deep learning activities, making it useful for complex operations on v3_2 that utilized the Dense Neural Network. The Jetson AGX Orin hardware accelerators were harnessed to speed data analysis, making it a viable candidate for the task.

2.2.8.6.Summary of Versions and Evolution

The Automated Real-Time CICFlowMeter Filtering IDS was developed in an iterative fashion because the authors wanted to improve network security. Every edition brought new features:

- **Version 1** established the core framework, using Tcpdump for packet capture and a Random Forest classifier for prediction.
- **Version 2** improved efficiency with concurrent packet capturing and processing, minimizing packet loss.
- **Version 3** refined the approach further by integrating flow management with Scapy, allowing for more precise and immediate processing of flows.
- **Version 3 Variant (v3_2)** represented a major shift by incorporating deep learning, which brought new capabilities in detecting non-linear and sophisticated attack patterns using TensorFlow.

Overcoming challenges while designing a real-time and network anomaly detection requires layering functions to the IDS system to make them an effective solution. The iteration we did above accomplishes just that. The system's design and implementation with NVIDIA Jetson AGX Orin as the development and deployment platform indicates it is a versatile tool for edge computing and thus a suitable tool for modern real-time network security.

3. RESULTS

3.1. Multiple Classification Results

3.1.1. Dense Neural Network (DNN) Results

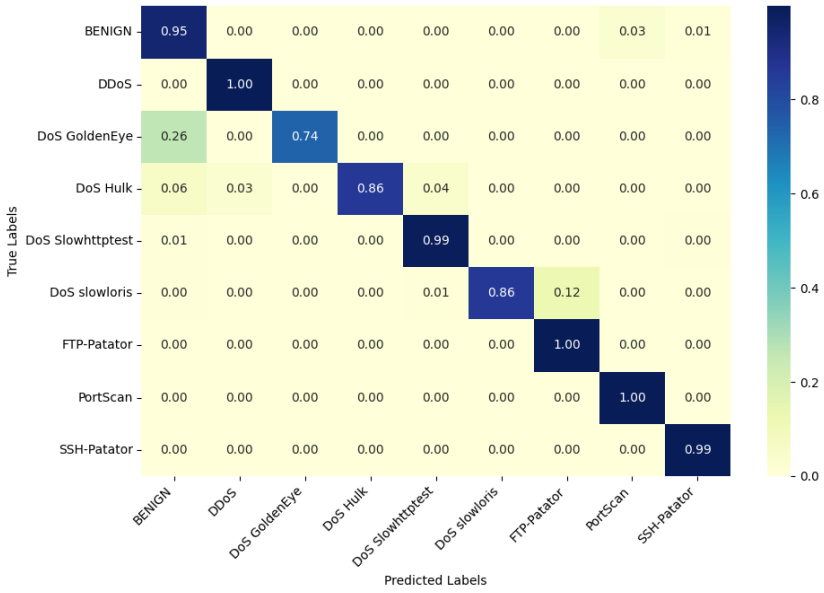
The performance of the Dense Neural Network (DNN) model was evaluated on the **multi-class datasets**, **MLCVE_clean** and **MLCVE_clean_dest**, using several configurations. These configurations varied based on **batch size**, **learning rate**, and whether **early stopping (ES)** was applied during training. For the purpose of evaluating the model, the **macro average** of **precision**, **recall**, and **F1-score** metrics was emphasized as it provides a balanced view of performance across all classes, irrespective of class imbalance.

The DNN results are presented below based on four key sets of experiments for each dataset. These results are explained individually before being compared to determine how different factors influenced the performance.

3.1.1.1.MLCVE_clean Dataset Results

The **MLCVE_clean** dataset was used in multiple training scenarios. Below, the macro average metrics for each configuration are presented, highlighting the model's performance on this dataset.

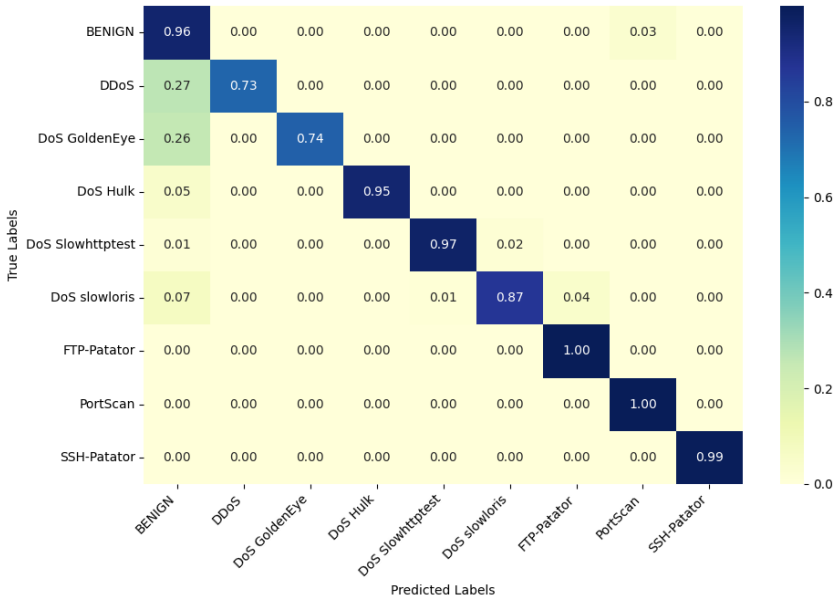
Figure 3.1 MLCVE_clean (B/S: 64, LR: 0.01, FI: Disabled, ES: Enabled):



The model presented in **Figure 3.1** achieved notable performance metrics, including a macro average precision of **0.848**, a macro average recall of **0.931**, and a macro average F1-score of **0.860**.

In this initial configuration, the **DNN** model showed good recall, indicating that the model was able to detect most of the attack classes well. However, the precision was slightly lower, meaning that there was a higher rate of false positives among predicted attack types. The macro average **F1-score** of **0.860** indicated a balanced performance, but there was room for improvement.

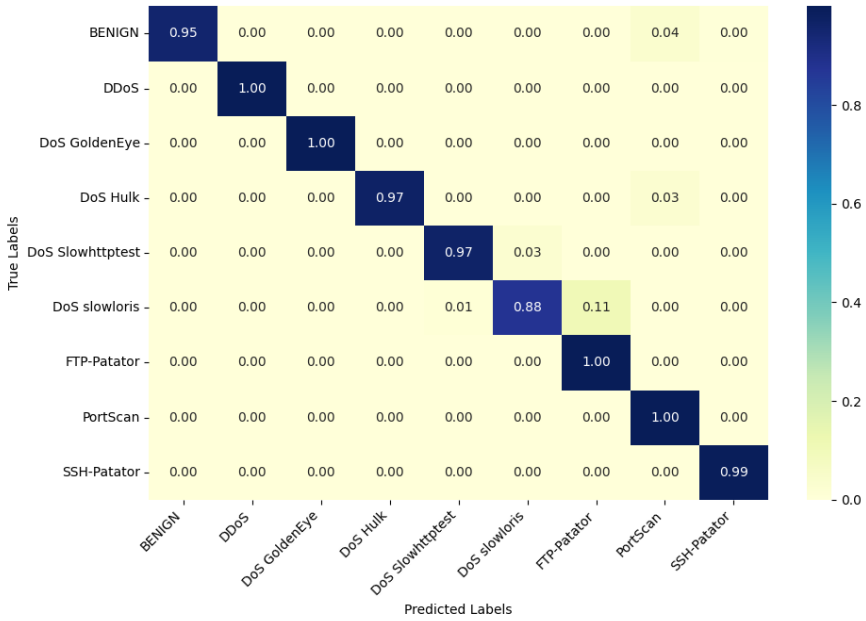
Figure 3.2 MLCVE_clean (B/S: 256, LR: 0.01, FI: Disabled, ES: Enabled):



The model presented in **Figure 3.2** achieved notable performance metrics, including a macro average precision of **0.933**, a macro average recall of **0.912**, and a macro average F1-score of **0.916.**

By increasing the **batch size** to **256**, there was a notable improvement in macro average **precision** to **0.933**. This indicates that larger batch sizes helped the model achieve more stable gradient estimates, leading to fewer false positives. The macro average **F1-score** improved to **0.916**, reflecting a better balance between precision and recall.

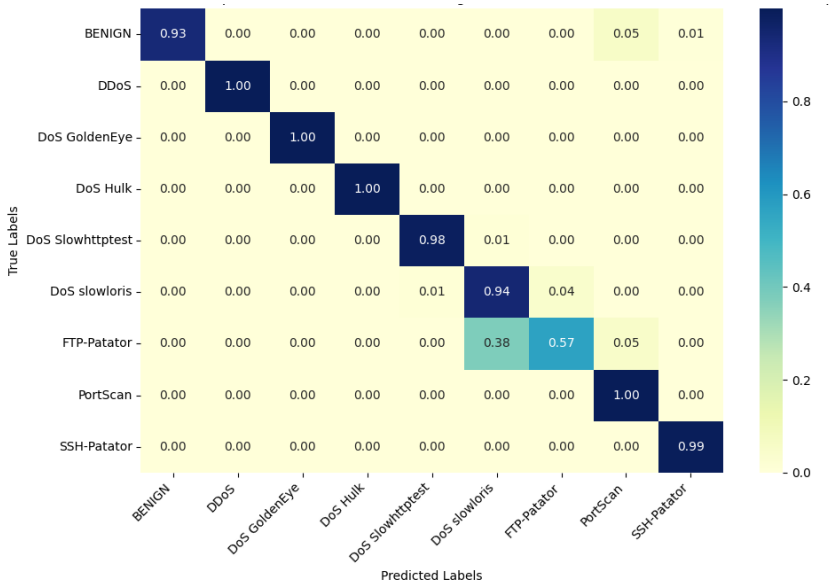
Figure 3.3 MLCVE_clean (B/S: 256, LR: 0.001, FI: Disabled, ES: Enabled):



The model presented in **Figure 3.3** achieved notable performance metrics, including a macro average precision of **0.957**, a macro average recall of **0.973**, and a macro average F1-score of **0.964.**

When the **learning rate** was reduced to **0.001**, the DNN achieved significantly higher macro average metrics across the board. The **F1-score** reached **0.964**, demonstrating that the model benefitted from slower, more precise learning, which led to better optimization.

Figure 3.4 MLCVE_clean (B/S: 256, LR: 0.001, FI: Disabled, No ES):



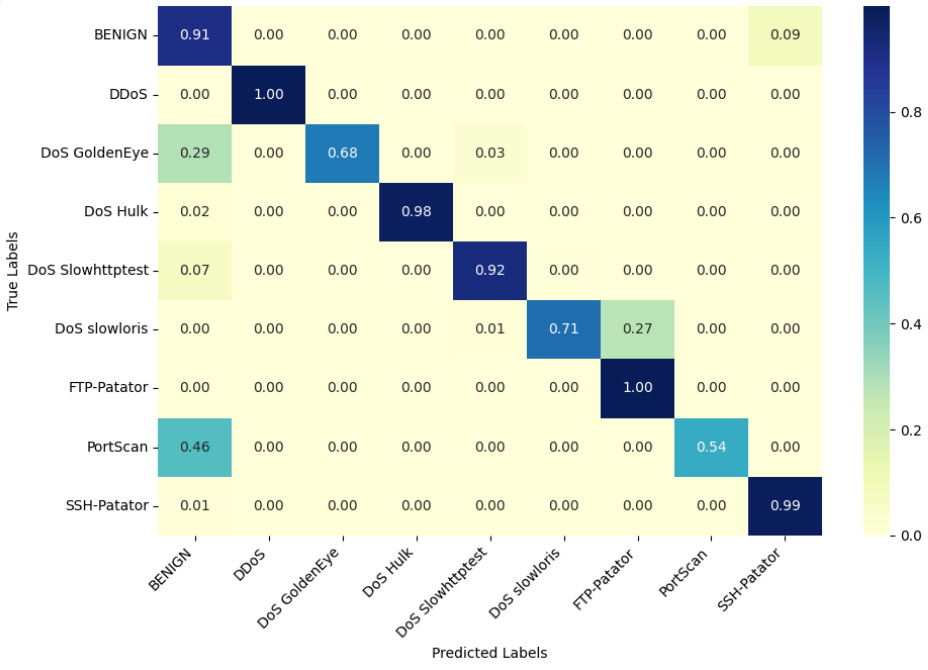
The model presented in **Figure 3.4** achieved notable performance metrics, including a macro average precision of **0.921**, a macro average recall of **0.936**, and a macro average F1-score of **0.919.**

When **early stopping** was disabled, the model performance declined slightly, with an **F1-score** of **0.919** compared to **0.964** when early stopping was enabled. This indicates that the early stopping mechanism played a significant role in preventing overfitting, leading to better generalization when training was halted at the optimal point.

3.1.1.2.MLCVE_clean_dest Dataset Results

The **MLCVE_clean_dest** dataset, which excluded the **Destination Port** feature, was also tested under similar training conditions to understand how the absence of this feature affected the model's classification ability.

Figure 3.5 MLCVE_clean_dest (B/S: 64, LR: 0.01, FI: Disabled, ES: Enabled):

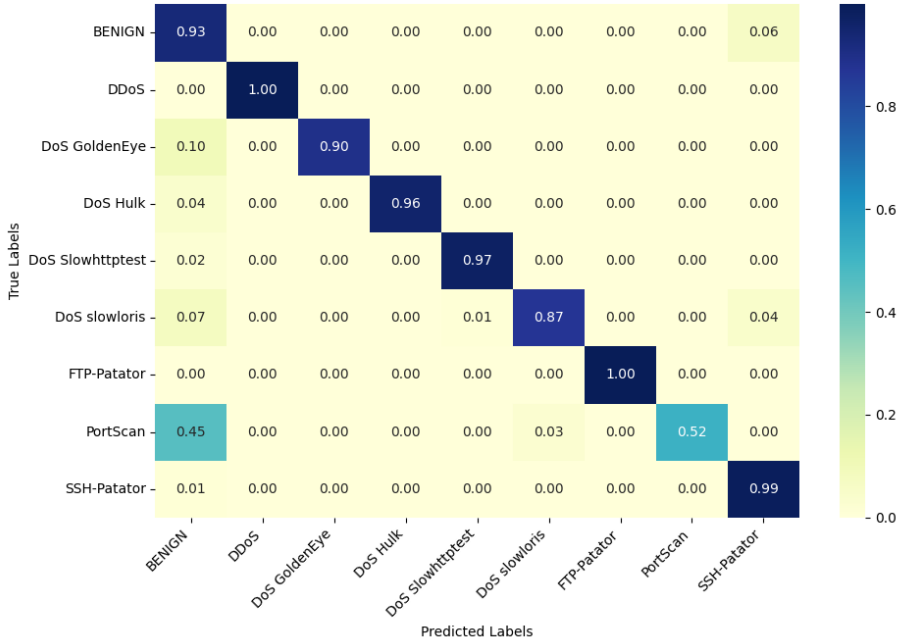


The model presented in **Figure 3.5** achieved notable performance metrics, including a macro average precision of **0.835**, a macro average recall of **0.858**, and a macro average F1-score of **0.811**.

For the initial configuration, the performance metrics for the **MLCVE_clean_dest** dataset were consistently lower than those for the **MLCVE_clean** dataset. The macro average **F1-**

score of **0.811** suggests that the removal of the **Destination Port** feature impacted the model's ability to effectively classify the network traffic, resulting in lower precision and recall.

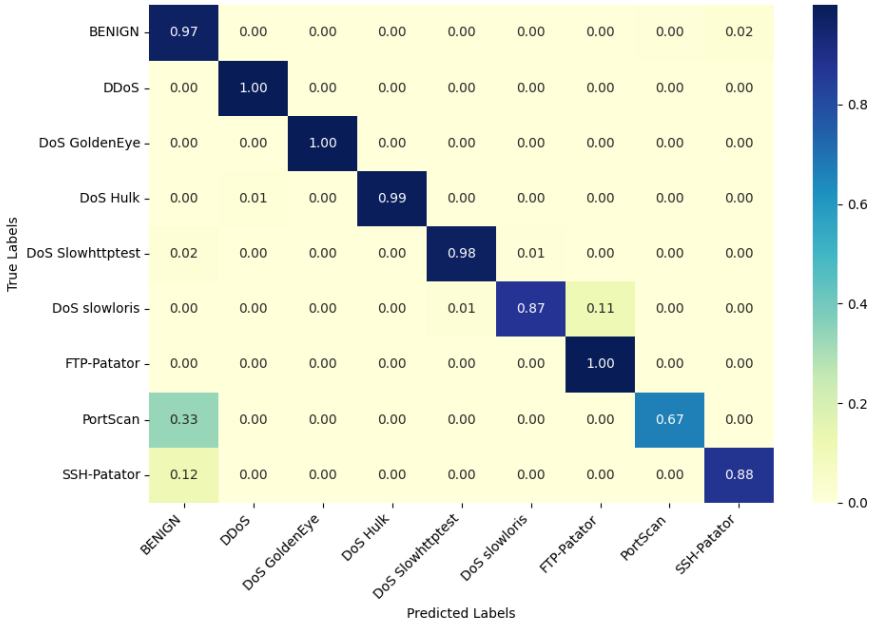
Figure 3.6 MLCVE_clean_dest (B/S: 256, LR: 0.01, FI: Disabled, ES: Enabled):



The model presented in **Figure 3.6** achieved notable performance metrics, including a macro average precision of **0.810**, a macro average recall of **0.903**, and a macro average F1-score of **0.820.**

Increasing the **batch size** to **256** improved the **recall** to **0.903**, indicating that the model could detect more attack types correctly. However, **precision** dropped slightly to **0.810**, suggesting an increased number of false positives. The **F1-score** was **0.820**, indicating a moderate balance between precision and recall.

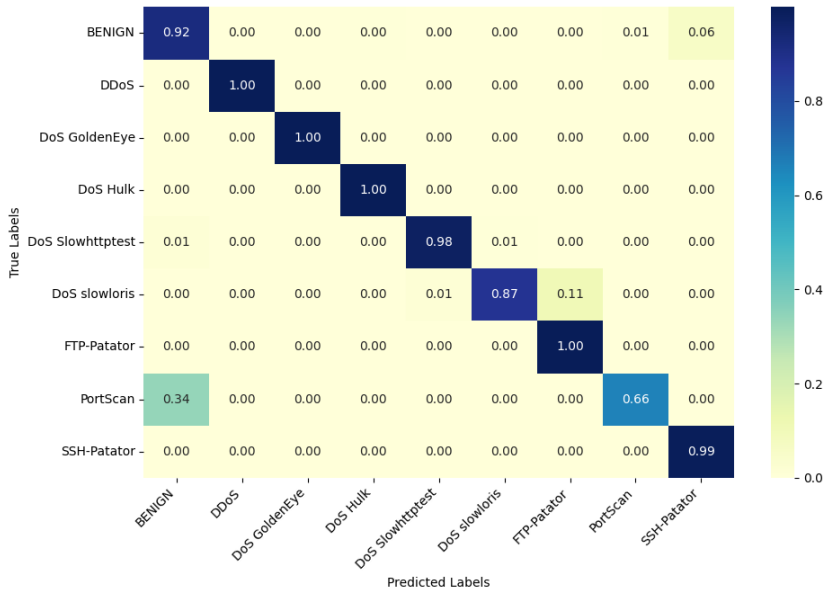
Figure 3.7 MLCVE_clean_dest (B/S: 256, LR: 0.001, FI: Disabled, ES: Enabled):



The model presented in **Figure 3.7** achieved notable performance metrics, including a macro average precision of **0.905**, a macro average recall of **0.928**, and a macro average F1-score of **0.909.**

Lowering the **learning rate** to **0.001** led to improvements in all metrics, with a **macro average F1-score** of **0.909**. This shows that even in the absence of the **Destination Port** feature, a lower learning rate allowed the model to optimize better and converge towards more accurate predictions.

Figure 3.8 MLCVE_clean_dest (B/S: 256, LR: 0.001, FI: Disabled, No ES):



The model presented in **Figure 3.8** achieved notable performance metrics, including a macro average precision of **0.869**, a macro average recall of **0.935**, and a macro average F1-score of **0.882.**

Training without **early stopping** resulted in a decline in performance, as seen with the **F1-score** of **0.882** compared to **0.909** when early stopping was enabled. This again suggests that early stopping is crucial to prevent overfitting and helps achieve better generalization in the final model. Now let's see these results from a comparative perspective.

3.1.1.3.Comparative Analysis of DNN Results

The next section compares DNN performance on the MLCVE_clean and MLCVE_clean_dest datasets, noting interesting insights obtained from varying training configurations such as batch size, learning rate and early stopping (ES).

A primary comparison was made between the performance of the DNN on the MLCVE_clean and MLCVE_clean_dest datasets. In this analysis, we looked at what would happen if we dropped the Destination Port feature, which was very important (according to feature importance analysis). **Table 3.1** summarizes the results of the two datasets under the same training conditions.

Table 3-1 Comparison of DNN Macro Average Metrics between MLCVE_clean and MLCVE_clean_dest

Dataset	Batch Size	Learning Rate	Early Stopping	Macro Precision	Macro Recall	Macro F1-Score
MLCVE_clean	64	0.01	Enabled	0.848	0.931	0.860
MLCVE_clean_dest	64	0.01	Enabled	0.835	0.858	0.811
MLCVE_clean	256	0.01	Enabled	0.933	0.912	0.916
MLCVE_clean_dest	256	0.01	Enabled	0.810	0.903	0.820

The analysis of the **Table 3.1**, the MLCVE_clean dataset is better than MLCVE_clean_dest in all configurations. Macro F1-score spikes for MLCVE_clean_dest in comparison with MLCVE_clean. Therefore, the Destination Port feature gave significant information than other features for classifying normal and attack traffic. Without this feature, the DNN could not classify traffic as effectively, especially the attack classes which might be heavily dependent on port information.

Another vital benchmark was based on the effect of different batch sizes. The influence of model convergence and stability was evaluated by training the DNN with batch sizes of 64 and 256. **Table 3.2** shows the macro average metrics for the DNN trained using varying batch sizes for each dataset.

Table 3-2 Comparison of DNN Macro Average Metrics for Different Batch Sizes

Dataset	Batch Size	Learning Rate	Early Stopping	Macro Precision	Macro Recall	Macro F1-Score
MLCVE_clean	64	0.01	Enabled	0.848	0.931	0.860
MLCVE_clean	256	0.01	Enabled	0.933	0.912	0.916
MLCVE_clean_dest	64	0.01	Enabled	0.835	0.858	0.811
MLCVE_clean_dest	256	0.01	Enabled	0.810	0.903	0.820

For both **MLCVE_clean** and **MLCVE_clean_dest**, an increase in the **batch size** from **64** to **256** resulted in improved **macro average precision**. For instance, precision for **MLCVE_clean** increased from **0.848** to **0.933**. However, recall dropped slightly, indicating that a larger batch size led to fewer false positives but at the potential cost of missing certain attack types.

The **learning rate** is a critical parameter that affects the model's convergence and overall performance. **Table 3.3** summarizes the performance metrics of the DNN trained with **learning rates of 0.01 and 0.001**.

Table 3-3 Comparison of DNN Macro Average Metrics for Different Learning Rates

Dataset	Batch Size	Learning Rate	Early Stopping	Macro Precision	Macro Recall	Macro F1-Score
MLCVE_clean	256	0.01	Enabled	0.933	0.912	0.916
MLCVE_clean	256	0.001	Enabled	0.957	0.973	0.964
MLCVE_clean_dest	256	0.01	Enabled	0.810	0.903	0.820
MLCVE_clean_dest	256	0.001	Enabled	0.905	0.928	0.909

Lowering the **learning rate** from **0.01** to **0.001** resulted in significant improvements across all metrics for both datasets. For **MLCVE_clean**, the **macro average F1-score** increased from **0.916** to **0.964**, indicating that a slower learning rate allowed the model to make more refined weight adjustments during training, reducing the risk of overshooting and leading to better optimization.

Finally, the impact of **early stopping (ES)** was examined to assess whether its use helped prevent overfitting and improved generalization. **Table 3.4** presents the metrics for models trained with and without early stopping enabled.

Table 3-4 Comparison of DNN Macro Average Metrics with and without Early Stopping

Dataset	Batch Size	Learning Rate	Early Stopping	Macro Precision	Macro Recall	Macro F1-Score
MLCVE_clean	256	0.001	Enabled	0.957	0.973	0.964
MLCVE_clean	256	0.001	Disabled	0.921	0.936	0.919
MLCVE_clean_dest	256	0.001	Enabled	0.905	0.928	0.909
MLCVE_clean_dest	256	0.001	Disabled	0.869	0.935	0.882

Use of early stopping improved the performance of the model on all metrics. The F1 score macro average was **0.964** with early stopping on **MLCVE_clean** and **0.919** when it was off. **MLCVE_clean_dest** was grown to the similar trend. The early stopping criterion was effective at preventing overfitting, meaning that the model did not train long enough to start memorizing noise within the data.

3.1.1.4.Summary of Comparative Insights

The analysis to compare DNN model trained on both datasets shows us some important things. Adding Destination Port feature in **MLCVE_clean** dataset led to better model performance as compared to **MLCVE_clean_dest** dataset. The result shows Destination Port distinguishes between normal and attack traffic and this information plays an important role in enhancing the classification capability of the model. Still, we must consider the

Destination Port vulnerability that is susceptible to manipulation as well.

Also, when we increased the batch sizes from **64** to **256**, the precision and F1-score improved in general which indicated a more stable training along with a fewer false positive. The increased batch sizes helped the model generalize better, which reduced false positives. However, there was also a small drop in recall indicating this came at the cost of the model's ability to pick up all attacks.

Decreasing the learning rate from **0.01** to **0.001** improved model performance across all metrics quite significantly. Using a slower learning rate would help adjust the model to better optimize the observed values of the image data. The end result was having more precise and accurate detection performance with more generalization capabilities.

The use of stopping early proved to be an important factor in the optimal performance of the model. Models that were trained with early stopping had higher F1 scores than those that were not The model's overfitting was avoided through early stopping, enhancing generalization on unseen data, which is critical for reliable network intrusion detection.

The best settings for MLCVE_clean were batch-size 256, learning rate 0.001, and early stopping on. This achieved a macro avg f1 score of 0.964. The second-best performance was obtained when the batch size was 256, the learning rate was 0.001 and early stopping was enabled yielding a macro average F1-score of 0.909 for MLCVE_clean_dest dataset. These configurations show that tuning the hyperparameters is essential for the best model result in both cases.

The result shows that the features pointed in this work are needed to be retained such as Destination Port and the hyper-parameters was needed to be tuned such as the batch size, learning rate, and early stopping, etc. Properly selecting a filter and tuning its parameters can greatly improve the capacity of the neural network to generalise. This could allow the neural network to detect and classify different types of traffic.

3.1.1.5.Alternative Architecture Results

The results for both datasets are shown in this section. It shows the results for multi-classification by layer architecture to optimizer preference.

Table 3-5 DNN Alternative Layer Architecture Results for MLCVE_clean

Dataset	Macro F1-Score	Accuracy %	Layer Configuration
MLCVE_clean	0.964	97.8	Original
MLCVE_clean	0.967	98.7	BottleNeck
MLCVE_clean	0.971	98.7	Sparse Wide
MLCVE_clean	0.973	98.7	Pyramid
MLCVE_clean	0.957	98.0	Reg. Batch Norm.
MLCVE_clean	0.970	98.7	Gradual Compression Reg.
MLCVE_clean	0.935	97.7	Single 32
MLCVE_clean	0.931	98.2	Single 64
MLCVE_clean	0.872	91.0	Single 128

The **Table 3.5** shows that Sparse Wide and Pyramid Layer configurations improve the performance of the model. As the difference between Pyramid and Sparse Wide are neglectable, Sparse Wide would be the more preferable choice because of its more cost-effective architecture compared to Pyramid architecture.

With Sparse Wide architecture, the accuracy improved over one percent. The f1 score improved almost one percent.

Table 3-6 Alternative Optimizer Architecture Results for MLCVE_clean

Dataset	Optimizer	Learning Rate	Accuracy (%)	Macro F1-Score	Additional Features
MLCVE_clean	Adam	0.001	98.7	0.971	-
MLCVE_clean	SGD	0.001	95.7	0.879	Momentum=0.9 (Accelerates convergence)
MLCVE_clean	RMSprop	0.001	87.4	0.812	Handles noisy gradients, useful for RNNs
MLCVE_clean	Adagrad	0.001	95.8	0.944	Adapts learning rate for each parameter, good for sparse data
MLCVE_clean	Adadelta	1.0	94.1	0.861	Addresses Adagrad's aggressive decay problem
MLCVE_clean	Nadam	0.001	90.7	0.847	Combines Adam and Nesterov momentum

Table 3.6 shows that using the Adam optimizer is still more effective compared with other alternatives. Next is MLCVE_clean_dest results.

Table 3-7 DNN Alternative Layer Architecture Results for MLCVE_clean_dest

Dataset	Macro F1-Score	Accuracy %	Layer Configuration
MLCVE_clean_dest	0.909	90.4	Original
MLCVE_clean_dest	0.883	89.0	BottleNeck
MLCVE_clean_dest	0.874	85.6	Sparse Wide
MLCVE_clean_dest	0.894	89.4	Gradual Compression Reg.
MLCVE_clean_dest	0.894	89.4	Reg. Batch Norm.
MLCVE_clean_dest	0.894	88.2	Pyramid
MLCVE_clean_dest	0.897	93.6	Single 128
MLCVE_clean_dest	0.905	93.9	Single 64
MLCVE_clean_dest	0.874	92.9	Single 32

From **Table 3.7**, it is apparent that when Destination Port info is not given, model performance drops. Testing different layer architectures for binary classification saw some improvements. Single hidden layer approach improved the performance compared to original 3 hidden layer architecture. While f1 score somewhat remined similar, accuracy increased by 3.5 percent. Next is the improvements done by optimizer preference in Sparse Wide model.

Table 3-8 Alternative Optimizer Architecture Results for MLCVE_clean_dest

Dataset	Optimizer	Learning Rate	Accuracy (%)	Macro F1-Score	Additional Features
MLCVE_clean_dest	Adam	0.001	90.4	0.909	-
MLCVE_clean_dest	Adadelta	1.0	95.5	0.910	Addresses Adagrad's aggressive decay problem
MLCVE_clean_dest	Nadam	0.001	94.0	0.896	Combines Adam and Nesterov momentum
MLCVE_clean_dest	Adagrad	0.001	91.9	0.882	Adapts learning rate for each parameter, good for sparse data
MLCVE_clean_dest	SGD	0.001	94.8	0.898	Momentum=0.9 (Accelerates convergence)
MLCVE_clean_dest	RMSprop	0.001	94.8	0.906	Handles noisy gradients, useful for RNNs

Table 3.8 shows that Adadelta optimizer improved the performance of the model compared to Adam optimizer.

Table 3-9 DNN Multiclassification Improvement

Dataset	Best Accuracy (%)	Best Macro F1-Score	Adjustment
MLCVE_clean	97.8	0.964	Original
MLCVE_clean	98.7	0.971	Optimizer: Adam Architecture: Sparse Wide
MLCVE_clean_dest	90.4	0.909	Original
MLCVE_clean_dest	95.5	0.910	Optimizer: Adadelta Architecture: Single 64

From **Table 3.9**, it can be understood that layer and optimizer adjustments improved the model performance. In the next part, we will look into the binary dataset results.

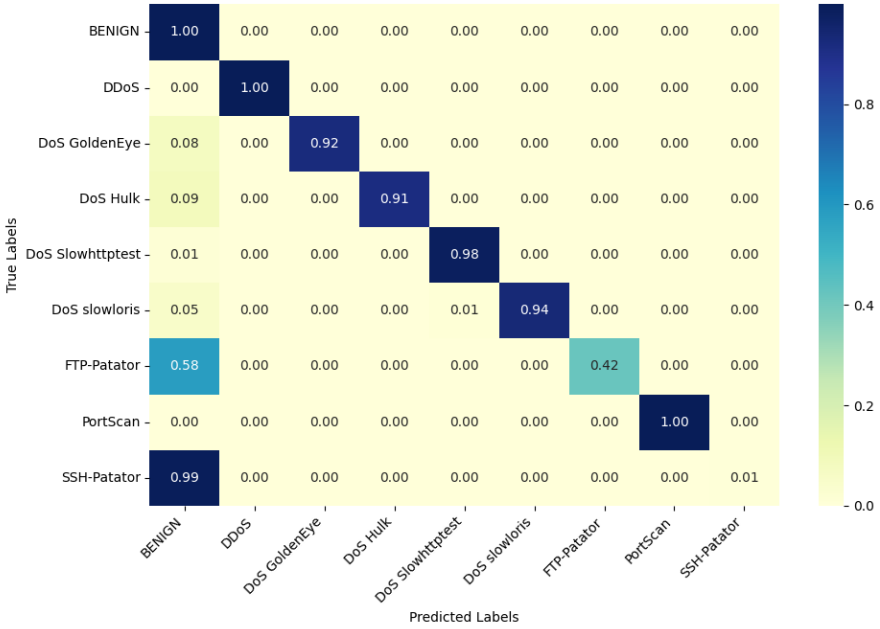
3.1.2. Random Forest Results

The **Random Forest (RF)** model was evaluated on the **MLCVE_clean** and **MLCVE_clean_dest** datasets across different configurations. These configurations varied based on the **maximum depth** of decision trees used in the model, allowing us to observe the effect of limiting model complexity. The results of each experiment are presented below, focusing on the **macro average** metrics of **precision**, **recall**, and **F1-score** to provide a balanced perspective across all classes, regardless of class imbalance.

3.1.2.1.MLCVE_clean Dataset Results

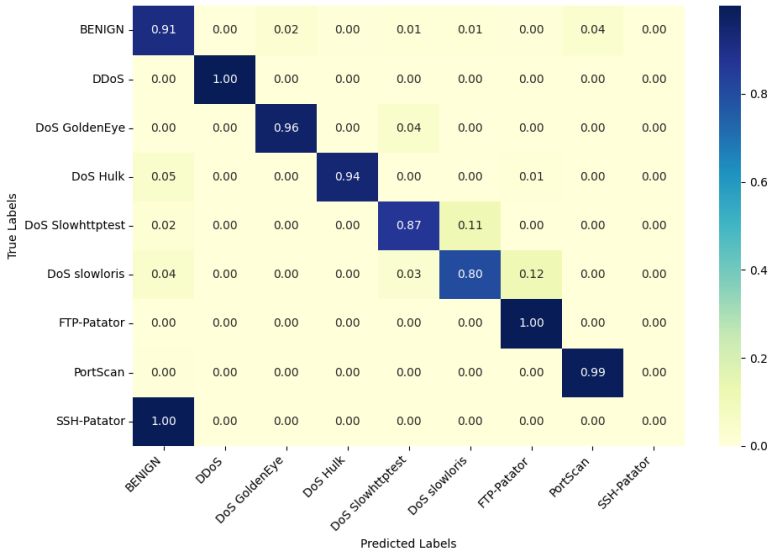
The **MLCVE_clean** dataset, which includes the **Destination Port** feature, was used in several training configurations with Random Forest.

Figure 3.9 MLCVE_clean - RandomForest Model (No Max Depth Limit):



The model presented in **Figure 3.9** achieved notable performance metrics. In the configuration without any restriction on the **maximum depth** of trees, the model achieved high precision across all classes, particularly with **macro average precision** at **0.982**. This suggests the model effectively identified attack classes with minimal false positives. However, the **recall** was lower (**0.797**), indicating that the model missed certain attack instances, resulting in a **macro average F1-score** of **0.821**. The overall accuracy was **95.7%**.

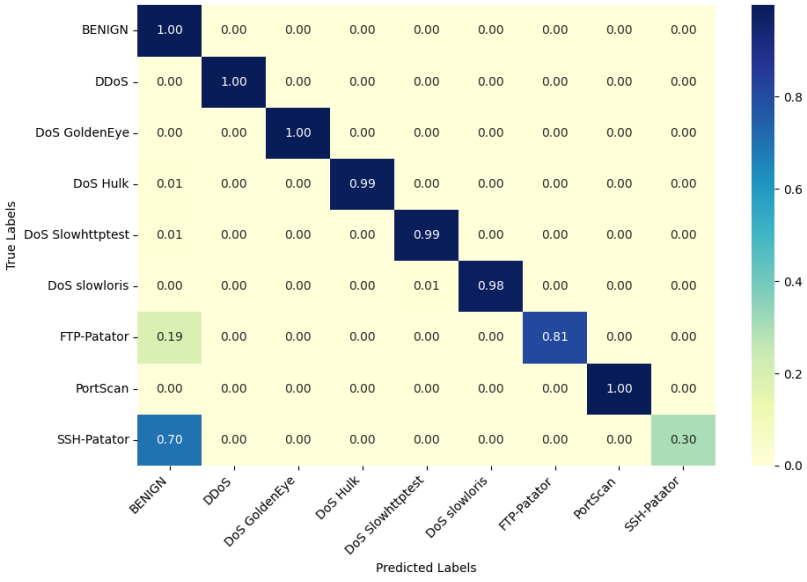
Figure 3.10 MLCVE_clean - RandomForest Model (Max Depth: 5):



The model presented in **Figure 3.10** achieved notable performance metrics, including a macro average precision of **0.816**, a macro average recall of **0.831**, a macro average F1-score of **0.755**, and an accuracy of **94.6%.**

Restricting the **maximum depth** to **5** significantly impacted the model's precision and recall. The model's **F1-score** dropped to **0.755**, reflecting reduced classification effectiveness. While simpler models (lower max depth) are typically less prone to overfitting, this reduction in complexity negatively impacted the model's ability to accurately classify attack classes.

Figure 3.11 MLCVE_clean - RandomForest Model (Max Depth: 10):



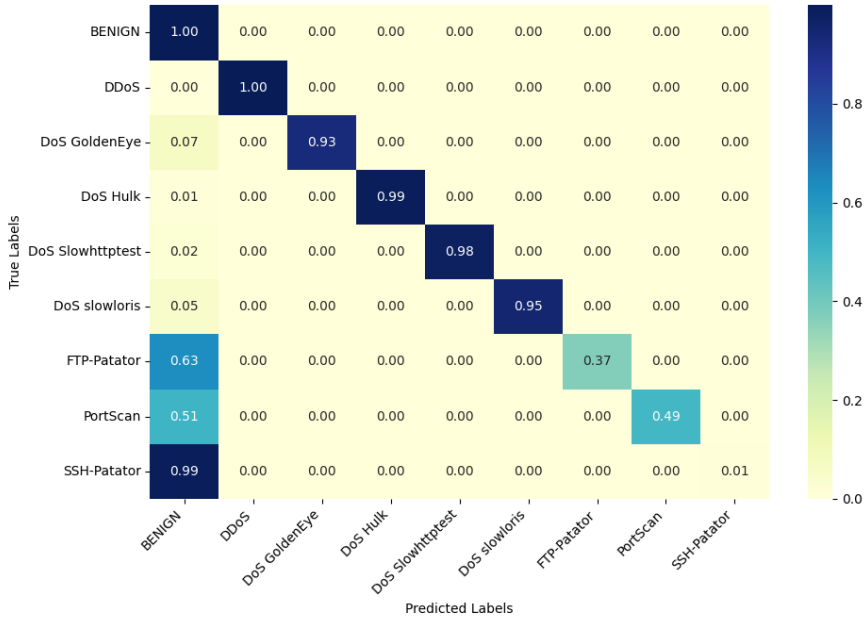
The model presented in **Figure 3.11** achieved notable performance metrics, including a macro average precision of **0.991**, a macro average recall of **0.896**, a macro average F1-score of **0.921**, and an accuracy of **98.6%**

With a **maximum depth** of **10**, the Random Forest model improved substantially across all metrics. The **macro average precision** of **0.991** and recall of **0.896** resulted in an **F1-score** of **0.921**. The accuracy also increased to **98.6%**, indicating a good balance between complexity and generalization.

3.1.2.2.MLCVE_clean_dest Dataset Results

The **MLCVE_clean_dest** dataset, in which the **Destination Port** feature was removed, was also evaluated using similar configurations.

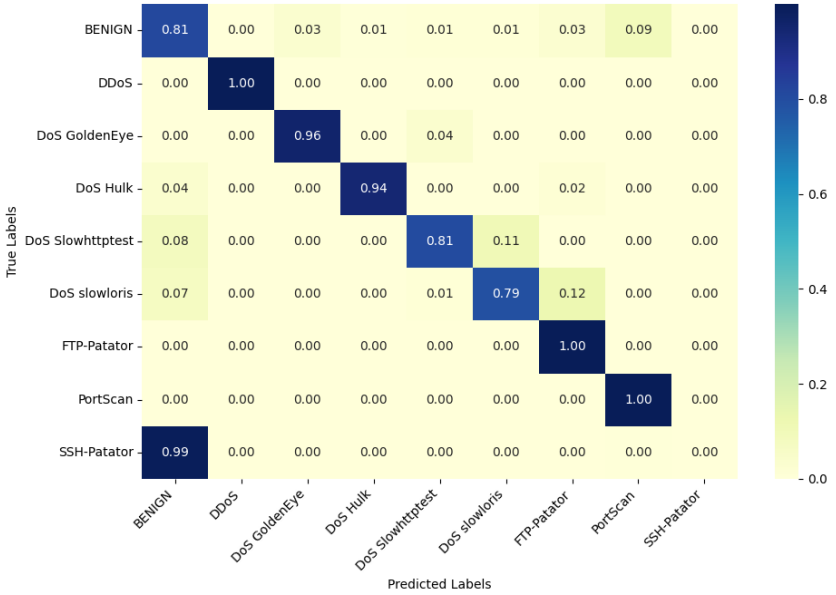
Figure 3.12 MLCVE_clean_dest - RandomForest Model (No Max Depth Limit):



The model presented in **Figure 3.12** achieved notable performance metrics, including a macro average precision of **0.957**, a macro average recall of **0.746**, a macro average F1-score of **0.766**, and an accuracy of **84.2%.**

Without a **maximum depth** limit, the model's **macro average F1-score** dropped to **0.766** when the **Destination Port** feature was removed. This suggests that the model struggled more without this feature, and the reduced **recall** (**0.746**) indicates that the model had difficulty identifying all instances of attack classes.

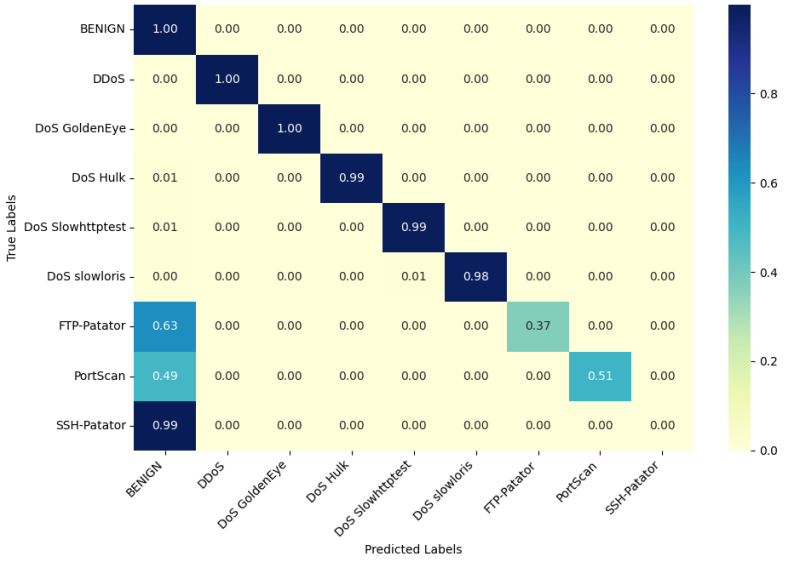
Figure 3.13 MLCVE_clean_dest - RandomForest Model (Max Depth: 5):



The model presented in **Figure 3.13** achieved notable performance metrics, including a macro average precision of **0.772**, a macro average recall of **0.813**, a macro average F1-score of **0.714**, and an accuracy of **92.2%**

When the **maximum depth** was restricted to **5**, the **macro average F1-score** dropped further to **0.714**, and the accuracy was **92.2%**. The loss of the **Destination Port** feature combined with reduced tree depth led to a substantial reduction in model performance.

Figure 3.14 MLCVE_clean_180_dest - RandomForest Model (Max Depth: 10):



The model presented in **Figure 3.14** achieved notable performance metrics, including a macro average precision of **0.947**, a macro average recall of **0.759**, a macro average F1-score of **0.768**, and an accuracy of **84.4%.**

Increasing the **maximum depth** to **10** improved the model's performance on **MLCVE_clean_dest**. However, the **macro average F1-score** of **0.768** was still lower compared to the **MLCVE_clean** dataset, indicating that the removal of the **Destination Port** feature negatively impacted overall classification effectiveness.

3.1.2.3.Comparative Analysis of Random Forest Results

A comparison of the **Random Forest** model results on **MLCVE_clean** and **MLCVE_clean_dest** datasets across different configurations reveals several important findings. **Table 3.10** summarizes the key metrics for each configuration.

Table 3-10 Comparison of Random Forest Macro Average Metrics for MLCVE_clean and MLCVE_clean_dest

Dataset	Max Depth	Macro Precision	Macro Recall	Macro F1-Score	Accuracy
MLCVE_clean	None	0.982	0.797	0.821	95.7%
MLCVE_clean_dest	None	0.957	0.746	0.766	84.2%
MLCVE_clean	5	0.816	0.831	0.755	94.6%
MLCVE_clean_dest	5	0.772	0.813	0.714	92.2%
MLCVE_clean	10	0.991	0.896	0.921	98.6%
MLCVE_clean_dest	10	0.947	0.759	0.768	84.4%

The comparison shows that MLCVE_clean is better than MLCVE_clean_dest in all configurations. This shows that it is the features related to the Destination Port which is critical to the model as removal of it led to large drop in the precision, recall and the F1 score.

For both datasets, we observed a decrease in all metrics after limiting the maximum depth of the trees to 5. This means the model was too simple to learn from the data for these stands. Raising the maximum depth to 10 allowed to have a decent trade-off between model complexity and performance. Particularly MLCVE_clean had a macro average F1-score of 0.921.

3.1.2.4.Summary of Comparative Insights

Performance of Random Forest Model on both the MLCVE_clean and MLCVE_clean_dest mentioned about key thing that is features and complexity of the model. The Destinacion Port feature removal impacted the model's classification performance badly and the precision, recall and F1-score of all configurations reduced. This shows that Destination Port allows the differentiation of applications that generate different flows.

Random Forest model performance was greatly impacted by limiting the maximum depth of trees used in it. Keeping the depth limited can avoid overfitting. But, it also decreases the ability to classify attack types in both datasets The 10-depth model is optimal, as it avoids overfitting while retaining most of model accuracy, especially for MLCVE_clean..

The highest performance was observed in the **MLCVE_clean** dataset using a **maximum depth of 10**, resulting in a **macro average F1-score of 0.921** and an **accuracy of 98.6%**. For **MLCVE_clean_dest**, the best configuration also used a **maximum depth of 10**, but the performance metrics, including an **F1-score of 0.768** and **accuracy of 84.4%**, were significantly lower, reinforcing the importance of the **Destination Port** feature.

3.1.3. Comparative Analysis of Dense Neural Network and Random Forest Results

The DNN model beats the RF model in both MLCVE_clean and MLCVE_clean_dest datasets. The DNN generalizes better across the various classes as seen from the higher macro average F1-scores and recall scores. The top DNN model with a macro average F1-score of 0.964 for the MLCVE_clean data set was trained with 256 batch size, 0.001 learning rate and early stopping. The Random Forest model had the best performance at 10 depth with the crop average F1 score of 0.921. When the Destination Port feature was removed in the MLCVE_clean_dest dataset, both the models performed poorly. But the DNN performed better with F1-score 0.909 while the Random Forest performed worse with maximum F1-

score 0.768. The results indicate that although Random Forest can handle feature information completely, DNN is more flexible and adaptable, making it the more preferred model to use in situations that might experience feature missingness.

Table 3-11 Best-Performing Model Configurations for Each Dataset

Dataset	Model Type	Configuration Details	Macro F1-Score	Accuracy
MLCVE_clean	Dense NN	B/S: 256, LR: 0.001, ES: Enabled	0.964	99.3%
MLCVE_clean	Random Forest	Max Depth: 10	0.921	98.6%
MLCVE_clean_dest	Dense NN	B/S: 256, LR: 0.001, ES: Enabled	0.909	95.8%
MLCVE_clean_dest	Random Forest	Max Depth: 10	0.768	84.4%

3.2. Binary Classification Results

3.2.1. DNN Results

The guessed binary classification results of Dense Neural Network (DNN) model considered two datasets, MLCVE_Binary and MLCVE_Binary_Dest. These datasets are created by combining the non-BENIGN classes to ABNORMAL of original multiclass datasets which are MLCVE and MLCVE_Dest.

The here here summarize different model configurations, including batch size, learning rate, and the effect of weight changes.

3.2.1.1.MLCVE_Binary Dataset Results

The results for the **MLCVE_Binary** dataset indicated that smaller batch sizes yielded slightly better accuracy and macro F1-scores. When a **batch size of 64** and a **learning rate of 0.001** were employed, the model achieved an **accuracy of 98.8%** and a **macro F1-score of 0.985**, as shown in **Table 3.12** below. The **ABNORMAL class F1-score** was consistently higher than that for the **BENIGN class**, suggesting a particular proficiency of the DNN model in identifying malicious activity.

Table 3-12 MLCVE_Binary Dataset Results

Batch Size	Learning Rate	Epochs	Weight Adjustment	Accuracy (%)	Macro F1-Score	ABNORMAL F1	BENIGN F1
64	0.001	250	Original	98.8	0.985	0.992	0.977
256	0.001	250	Original	98.7	0.983	0.991	0.975
256	0.01	250	Original	98.7	0.983	0.990	0.976

The findings indicate that smaller batch sizes allow the model to capture the variations in the data more accurately ideal for the effective detection of both BENIGN and ABNORMAL traffic. Given the rich feature set of MLCVE_Binary, the high overall performance suggests that it can be used for binary classification with a DNN model without any significant weight changes.

3.2.1.2.MLCVE_Binary_Dest Dataset Results

When the **Destination Port** feature was removed, a significant decline in model performance was observed. The initial accuracy was **72.9%**, reflecting the importance of this feature in distinguishing between benign and malicious traffic. To compensate for this, different weight adjustments were applied, which improved the model's ability to classify correctly. With **Weight Adjustment W4**, the accuracy increased to **88.5%**, and the **macro F1-score** improved to **0.866**.

Table 3-13 MLCVE_Binary_Dest Dataset Results

Batch Size	Learning Rate	Epochs	Weight Adjustment	Accuracy (%)	Macro F1-Score	ABNORMAL F1	BENIGN F1
64	0.001	250	Original	72.9	0.716	0.734	0.698
64	0.001	250	W2	84.4	0.824	0.847	0.801
64	0.001	250	W3	86.7	0.848	0.863	0.833
64	0.001	250	W4	88.5	0.866	0.881	0.850

The results indicate that **weight adjustment** is an effective method to compensate for missing critical features, especially in binary classification tasks. The performance gains achieved using **W4** show that rebalancing class weights can significantly improve classification of both ABNORMAL and BENIGN flows, which would otherwise be impaired by the absence of key distinguishing attributes.

3.2.1.3.Comparative Analysis of DNN Results

The comparative analysis between the two datasets reveals the substantial impact of the **Destination Port** feature on classification performance. The **MLCVE_Binary** dataset showed a superior accuracy of **98.8%**, highlighting its comprehensive nature for anomaly detection. In contrast, the **MLCVE_Binary_Dest** dataset, even with optimized weight adjustments, could only reach an accuracy of **88.5%**.

Dataset	Best Accuracy (%)	Best Macro F1-Score	Optimal Weight Adjustment
MLCVE_Binary	98.8	0.985	Original
MLCVE_Binary_Dest	88.5	0.866	W4

These differences underscore the necessity of feature completeness to train good IDS models. The Destination Port is important for differentiating traffic type. Without this, additional weight tuning is needed to get reasonable performance.

3.2.1.4.Alternative Architecture Binary Results

This section is about improvements on binary classification by architecture alternatives. We will look into the improvements for both datasets.

Table 3-14 DNN Alternative Layer Architecture Results for MLCVE_binary

Dataset	Best Accuracy (%)	Best Macro F1-Score	Configuration
MLCVE_Binary	98.8	0.985	Original
MLCVE_Binary	98.7	0.983	BottleNeck
MLCVE_Binary	98.8	0.984	Sparse Wide
MLCVE_Binary	98.7	0.983	Pyramid
MLCVE_Binary	97.9	0.972	Reg. Batch Norm.
MLCVE_Binary	98.9	0.986	Gradual Compression Reg.
MLCVE_Binary	95.4	0.936	Single 128
MLCVE_Binary	95.4	0.937	Single 64
MLCVE_Binary	93.1	0.901	Single 32

The **Table 3.14** shows that Gradual Compression Reg and Original Layer configurations give almost the same performance for the model. As the difference between Gradual Compression Reg and Original Layer configurations are neglectable Original would be the more preferable choice because of its more cost-effective architecture compared to Gradual Compression Reg architecture.

Table 3-15 Alternative Optimizer Architecture Results for MLCVE_binary

Dataset	Optimizer	Learning Rate	Accuracy (%)	Macro F1-Score	Additional Features
MLCVE_Binary	Adam	0.001	98.8	0.985	-
MLCVE_Binary	Adadelta	1.0	98.9	0.986	Addresses Adagrad's aggressive decay problem
MLCVE_Binary	Nadam	0.001	98.6	0.982	Combines Adam and Nesterov momentum
MLCVE_Binary	SGD	0.001	98.9	0.986	Momentum=0.9 (Accelerates convergence)
MLCVE_Binary	Adagrad	0.001	98.6	0.982	Adapts learning rate for each parameter, good for sparse data
MLCVE_Binary	RMSprop	0.001	98.6	0.982	Handles noisy gradients, useful for RNNs

Table 3.15 shows that using the Adam optimizer is still more effective compared with other alternatives. Next is MLCVE_binary_dest results.

Table 3-16 DNN Alternative Layer Architecture Results for MLCVE_binary_dest

Dataset	Best Accuracy (%)	Best Macro F1-Score	Configuration
MLCVE_Binary_Dest	91.7	0.866	Original
MLCVE_Binary_Dest	83.4	0.815	BottleNeck
MLCVE_Binary_Dest	85.6	0.784	Sparse Wide
MLCVE_Binary_Dest	83.2	0.718	Pyramid
MLCVE_Binary_Dest	85.6	0.837	Reg. Batch Norm.
MLCVE_Binary_Dest	87.3	0.847	Gradual Compression Reg.
MLCVE_Binary_Dest	87.3	0.853	Single 128
MLCVE_Binary_Dest	87.3	0.848	Single 64
MLCVE_Binary_Dest	89.0	0.870	Single 32
MLCVE_Binary_Dest	86.7	0.846	Single 16

It is apparent in **Table 3.16** that when Destination Port info is not given, model performance drops. Testing different layer architectures for binary classification saw some changes. Still, original architecture remains better. Next is the improvements done by optimizer preference in original model.

Table 3-17 Alternative Optimizer Architecture Results for MLCVE_binary_dest

Dataset	Optimizer	Learning Rate	Accuracy (%)	Macro F1-Score	Additional Features
MLCVE_Binary_Dest	Adam	0.001	91.7	0.866	-
MLCVE_Binary_Dest	Adadelta	1.0	95.1	0.939	Addresses Adagrad's aggressive decay problem
MLCVE_Binary_Dest	RMSprop	0.001	92.8	0.912	Handles noisy gradients, useful for RNNs
MLCVE_Binary_Dest	Nadam	0.001	92.8	0.912	Combines Adam and Nesterov momentum
MLCVE_Binary_Dest	SGD	0.001	88.9	0.870	Momentum=0.9 (Accelerates convergence)
MLCVE_Binary_Dest	Adagrad	0.001	86.6	0.847	Adapts learning rate for each parameter, good for sparse data

Table 3.17 shows that Adadelta optimizer improved the performance of the model compared to Adam optimizer in here as well, just like in multi-classification.

Table 3-18 DNN Binary Classification Improvement

Dataset	Best Accuracy (%)	Best Macro F1-Score	Adjustment
MLCVE_binary	98.8	0.985	Original
MLCVE_binary	98.9	0.986	Optimizer: Adadelta or SGD Architecture: Original
MLCVE_binary_dest	90.4	0.909	Original
MLCVE_binary_dest	95.1	0.939	Optimizer: Adadelta Architecture: Original

From **Table 3.18**, it can be understood that layer and optimizer adjustments improved the model performance. In the next part, we will look into the binary dataset results.

3.2.2. Random Forest Results

Similar to the DNN results, Random Forest models were also tested for binary classification across both datasets. The following sections summarize the key findings.

3.2.2.1.MLCVE_Binary Dataset Results

The **MLCVE_Binary** dataset results for the Random Forest model indicated a high level of performance, with an **accuracy of 97.6%** and a **macro F1-score of 0.970**. The results, presented in below, demonstrate the effectiveness of this traditional machine learning approach in detecting anomalies in a feature-rich dataset.

Max Depth	Weight Adjustment	Accuracy (%)	Macro Score	F1- ABNORMAL F1	BENIGN F1
10	Original	97.6	0.970	0.984	0.955

The **ABNORMAL class F1-score** again outperformed the **BENIGN class**, suggesting a stronger focus on detecting malicious behavior. This high performance, however, comes at the cost of lower interpretability when compared to deep learning models.

3.2.2.2.MLCVE_Binary_Dest Dataset Results

When the **Destination Port** feature was excluded, the performance of the Random Forest model also experienced a significant decline, as shown in Table 3.19. The initial **accuracy was 69.3%**, but by applying **Weight Adjustment W3**, the model's performance improved to **85.4%** with a **macro F1-score of 0.835**.

Table 3-19 MLCVE_Binary_Dest W3 Results

Max Depth	Weight Adjustment	Accuracy (%)	Macro F1-Score	ABNORMAL F1	BENIGN F1
5	Original	69.3	0.684	0.702	0.665
10	Original	69.6	0.686	0.704	0.668
10	W3	85.4	0.835	0.854	0.816

The W3 Weight Adjustment strategy worked very well to reduce performance loss caused by omitting Destination Port. So rebalancing for traditional models becomes important when a key feature is missing.

3.2.2.3 Comparative Analysis of Random Forest Results

The **MLCVE_Binary** dataset continued to show strong results, achieving a **best accuracy of 97.6%**. However, the **MLCVE_Binary_Dest** dataset showed the model's reliance on the **Destination Port** feature. The final performance of **85.4%** indicates that the dataset without **Destination Port** is less capable of accurately classifying network events using Random Forest.

Dataset	Best Accuracy (%)	Best Macro F1-Score	Optimal Weight Adjustment
MLCVE_Binary	97.6	0.970	Original
MLCVE_Binary_Dest	85.4	0.835	W3

These results further confirm the critical role that certain features play in the accuracy of IDS models. In scenarios where feature reduction is necessary, rebalancing techniques can mitigate the loss in classification ability.

3.2.3 Comparative Analysis of DNN and Random Forest for Binary Classification

A comparative analysis of the **Dense Neural Network (DNN)** and **Random Forest** models provides a broader understanding of their capabilities with respect to the two binary classification datasets.

Table 3-20 Comparative Analysis of DNN and Random Forest for Binary Classification

Dataset	Model	Best Accuracy (%)	Best Macro F1-Score	Optimal Weight Adjustment
MLCVE_Binary	DNN	98.8	0.985	Original
MLCVE_Binary	RandomForest	97.6	0.970	Original
MLCVE_Binary_Dest	DNN	88.5	0.866	W4
MLCVE_Binary_Dest	RandomForest	85.4	0.835	W3

The DNN model was superior to the Random Forest in both datasets. For the MLCVE_Binary dataset, the accuracy of DNN was 98.8% and that of Random Forest was 97.6%, both of which are very good. The dataset's being rich in features may allow the DNN more flexibility giving it a slight advantage.

For the MLCVE_Binary_Dest dataset, if the Destination Port feature is not included, then both models perform worse. The DNN with Weight Adjustment W4, achieved an accuracy of 88.5% and was better than Random Forest, which got 85.4% with weigh adjustment W3.

These findings show that deep learning models can remain effective even when features are cut. All they need is proper adjustment. Even though Random Forest was still effective, it couldn't adapt as much as the other algorithms did, which shows its limitations without those features. They showed that in IDS applications where feature completeness cannot be guaranteed DNN model can be a more robust solution.

3.3.Model Training on NVIDIA Jetson AGX Orin and Comparative Analysis

Through this study, we trained the Deep Neural Network (DNN) model on different platforms, specifically the NVIDIA A100, L4, T4 GPUs on Google Colab and the NVIDIA Jetson AGX Orin device. The aim to see if Jetson Orin, which is low-resource and portable hardware, can provide performance as well as accuracy on scale with larger, more powerful compute units when used to train a model for an IDS application. This analysis is very important when we think about using auto IDS anywhere with limited computational capacity.

3.3.1. Training Time Comparison

The DNN model training times for each of the platforms are summarized in the **Table 3.21** below:

Table 3-21 DNN model training times for each different platforms

Platform	Training Time (seconds)	Accuracy
NVIDIA A100 (Google Colab)	229.72	0.96
NVIDIA L4 (Google Colab)	223.90	0.96
NVIDIA T4 (Google Colab)	217.12	0.96
NVIDIA Jetson AGX Orin	2311.05	0.96

As we see in the **Table 3.21**, the Jetson Orin takes a long time to finish training (2311.05 seconds), while the high-performance GPUs, a stat of A100, L4 and T4 between 217-230 seconds. It should be noted, however, the accuracy values that were achieved across all platforms were in similar range (0.96 to 0.97). In other words, we can say that because Jetson Orin has less computational power, training it will take more time, but the results will be similar to strong GPUs.

3.3.2. Model Training Insights

Above graphs indicate the training performed on the Jetson AGX Orin along with accuracy and performance statistics. The regular accuracy of trained model on all platforms indicate that the generalization power of the trained model does not get affected when the platform

changes, as long as the model architecture, hyperparameters and training dataset remain constant.

This comparison means that you can train this model on A100 or L4, which speeds up training. The trained model can be saved and loaded onto Jetson AGX Orin for inference and deployment. It is an effective approach, especially in real life, when training is expensive but deployment should be light and energy-efficient.

3.3.3. Deployment on Jetson AGX Orin

The Jetson AGX Orin was chosen for use because it is well suited to edge AI. It has quite a lot of processing power, and it's small and efficient. The Orin adapts for inference, which is the major function for a real-time intrusion detection system, even though it is slower at training. A practical suggestion deriving from the study results is to train the models using high-performance computing resources and to then deploy them on the Jetson AGX Orin to efficiently monitor and detect real-time networks intrusion.

Training on powerful GPUs and deploying on a resource-efficient platform like Orin fits into the overall strategy of developing a practical and scalable intrusion detection system that remains sufficiently accurate and reliable in real-time.

3.3.3.1.Online Results Analysis from Embedded System

The prediction results from embedded system were analyzed if the predictions had any inconsistency. 4 attacks were detected in the first 800 flows. These attacks were 3 "DoS Slowloris" and 1 "DoS Hulk" attacks. When "DoS Slowloris" attacks were analyzed, it was determined that they were false positive. All three of them were flows with one packet. The common point of these three flows were that their information related to packet length and forward, backward packet movements were identical and their destination port were same. Other than that, there were no indication to support that these flows were "DoS Slowloris" such as keeping server busy as long as possible without ending requests.

On the other hand, when the single "DoS Hulk" prediction was analyzed, it was noticed that it had some striking similarities to the characteristic of the "DoS Hulk" in the training dataset of the model. Such as Flow Duration, Flow and Inter-Arrival Times and Idle Metrics. However, when the individual flow was extracted from the captured traffic, it had only two

packets. It was inconsistent with the buildup of “DoS Hulk” attacks so it determined it was also false positive.

Then same traffic was tested on binary model. The binary model detected only one abnormal flow within the traffic in the first 800 flows. When the flow was inspected, it was the same one from before that was determined as “DoS Hulk” by multi-classification model. Hence, the aforementioned flow having a correlating data with training data was the reason for false positive for both models. However, it can be argued that binary model had more consistency compared to multi-classification model as it did not false positive other three flows in multi-classification.

3.3.4. Conclusion

To conclude, Jetson AGX Orin can be a reasonable off-the-shelf portable solution for real-time IDS deployment, which is not most ideal for model training due to slower processing. The DNN might be trained on an NVIDIA A100, L4, or T4 before being deployed on Orin for inference in this case to offer a balance between computational efficiency and effective real-time security monitoring. Therefore, our Automated Real Time CICFlowMeter Filtering IDS can perform well without being limited by the processing power of the edge hardware.

4. DISCUSSION

The rapidly evolving digital world has significantly increased the need for advanced security measures to protect our data. This thesis aimed to create an automated real-time intrusion detection system (IDS). It followed steps of methodical process of dataset selection, preprocessing, and model training. It was followed by an embedded system integration for real-time deployment. The main focus of this study was by using state-of-art machine learning techniques, Dense Neural Networks (DNN) and Random Forest (RF), and combining them with practical computational hardware, such as the NVIDIA Jetson AGX Orin to detect malicious network activities.

Dataset selected was CICIDS2017 dataset. It was selected because of its diversity in capturing real-world network behavior. It also included dataset having 15 different classes of network traffic. An important aspect of this work lies in detailed pre-processing of this dataset. It allowed to create multiple versions, both with and without destination port information. As well as binary and multi-class, 9 classes, versions were created. This allowed for a detailed evaluation of the impact different features on model performance.

The finding for models shows that the presence of destination port plays an important role model performance. Following information are based on macro F1 scores. This measurement is a more correct measurement of the model as it evaluates all the classes in a balanced manner. For multi-class classification, the models achieved an average accuracy of 96.4%. When destination port information was removed from, it dropped to 91.0%. Similarly, in binary classification the accuracy was 86.6% while destination port information was not included, compared to 98.5% when included. These results highlight the impact of destination port information as a feature in differentiating between different types of network activity. Later when alternative architecture approaches were tried. As a result, multiclassification result was improved to 95.5% and binary classification result from 95.1% when destination info is not given. The Dense Neural Network models were able to capture complex attack patterns. They contributed to the higher accuracy observed, especially in the multi-class tasks.

However, the study also points out some limitations. First, although CICIDS2017 is a great dataset compared to older datasets like KDDCup99 and NSL-KDD, it doesn't cover all emerging attack types. Second, even though the inclusion of destination port data improved model accuracy, this approach also has risks. It may lead to potential overfitting when specific ports are correlated with certain types of attacks which is something that could be exploited by adversaries. Lastly, the usage of models on the NVIDIA Jetson AGX Orin. While it is effective, it revealed performance differences compared to high-performance cloud platforms, with training runtimes being considerably longer. This highlights the trade-off between using accessible edge devices and the need for high-speed detection in large-scale networks in matters of model training.

There were also some unexpected results. The high consistency in accuracy metrics across different deployment platforms, such as cloud-based GPUs (A100, L4, T4) and the NVIDIA Jetson AGX Orin. This suggests that for these datasets, model quality is primarily influenced by the dataset and feature quality rather than the underlying computational platform. Such findings open new opportunities for deploying IDS models on edge devices without substantial compromises in detection performance.

These findings align with other research efforts that points out the importance of dataset preprocessing and feature engineering in developing effective IDS models. The use of advanced machine learning algorithms, such as DNNs, has proven effective for multi-class classification in network security, similar to previous studies. However, this study's inclusion of both binary and multi-class datasets with different configurations of feature sets, allows for a better understanding of how machine learning models behave in different scenarios. It also reveals how different types of network features, such as destination ports, can influence model performance in a real-world application. However, integrating the trained models into an embedded system addresses the application and integration issues of models trained in previous researches into the real world.

Further research is recommended to address the limitations observed in this study. Specifically, using semi-supervised or unsupervised learning techniques could help improve the adaptability of the IDS to detect new, unseen attack types.

This thesis has addressed the challenges of real-time intrusion detection. The approach of combining sophisticated machine learning techniques, robust dataset preprocessing, and real-time deployment on edge hardware provides a foundation for further development in IDS research. Future iterations could focus on improving model efficiency and expanding the range of detectable threats to ensure the IDS remains effective in rapidly changing cybersecurity environments.

5. CONCLUSION AND FUTURE WORK

This thesis used machine learning in an effort to deal with the increasing demand for advanced IDS. To achieve this, we employed the Dense Neural Networks (DNN) and Random Forest (RF) models. The CICIDS2017 dataset is utilized to build a complete pipeline, which includes Preprocessing, Feature Selection, Model Training & Real-time deployment. Our detailed results show that if the destination port information is included in the model the macro F1 accuracy achieved will be over 97% on multi-class and 98.5% on binary. The accuracy achieved would be 91% and 86.6% without inclusion while with layer architecture improvements, it increased to 91.1% and 93.9%. Consequently, this highlights the quality of the dataset and careful feature selection have a notable influence on the efficiency of IDS.

A major contribution of this research was the deployment of the IDS system on NVIDIA Jetson AGX Orin. This demonstrated that sophisticated detection models could run effectively on embedded systems. However, training on the Orin took much longer than on high-power cloud environments like Google Colab. This led us to recommend training models in cloud environments. Then deploying them on edge devices like Orin. This approach helps achieve efficient performance while balancing practical constraints, such as computational resources and cost.

Despite these advances, some limitations remain. The dataset has inherent constraints, and certain features like destination ports can be manipulated, affecting detection. Future work should explore semi-supervised or unsupervised learning methods to detect unknown threats. There is also room for improvement by creating lightweight models through pruning or using techniques like federated learning. These approaches can enhance scalability, make the system more suitable for edge devices, and improve adaptability. This work contributes towards developing IDS solutions that are efficient, resilient, and adaptable for real-time application on cybersecurity.

6. APPENDIX

APPENDIX A: Explanation of Features in the CICIDS2017 Dataset (Original Dataset)

Feature Category	Feature Name (CICIDS2017)	Feature Name (CICFlowMeter)	Description
Basic Flow Features	Flow ID	Flow ID	Unique identifier for a flow.
Basic Flow Features	Source IP	Source IP	IP address of the source.
Basic Flow Features	Source Port	Source Port	Port number used by the source.
Basic Flow Features	Destination IP	Dst IP	IP address of the destination.
Basic Flow Features	Destination Port	Dst Port	Port number used by the destination.
Basic Flow Features	Protocol	Protocol	Protocol used in the connection (e.g., TCP, UDP, ICMP).
Basic Flow Features	Timestamp	Timestamp	Time when the flow was captured.
Basic Flow Features	Flow Duration	Flow Duration	Duration of the flow in microseconds.
Packet-Level Features	Total Fwd Packets	Tot Fwd Pkts	Total number of packets sent from the source to the destination.
Packet-Level Features	Total Backward Packets	Tot Bwd Pkts	Total number of packets sent from the destination to the source.
Packet-Level Features	Total Length of Fwd Packets	TotLen Fwd Pkts	Total size (in bytes) of all packets sent from source to destination.
Packet-Level Features	Total Length of Bwd Packets	TotLen Bwd Pkts	Total size (in bytes) of all packets sent from destination to source.
Packet-Level Features	Fwd Packet Length Max	Fwd Pkt Len Max	Maximum packet size observed in the forward direction.
Packet-Level Features	Fwd Packet Length Min	Fwd Pkt Len Min	Minimum packet size observed in the forward direction.

Feature Category	Feature Name (CICIDS2017)	Feature Name (CICFlowMeter)	Description
Packet-Level Features	Fwd Packet Length Mean	Fwd Pkt Len Mean	Average packet size in the forward direction.
Packet-Level Features	Fwd Packet Length Std	Fwd Pkt Len Std	Standard deviation of packet sizes in the forward direction.
Packet-Level Features	Bwd Packet Length Max	Bwd Pkt Len Max	Maximum packet size observed in the backward direction.
Packet-Level Features	Bwd Packet Length Min	Bwd Pkt Len Min	Minimum packet size observed in the backward direction.
Packet-Level Features	Bwd Packet Length Mean	Bwd Pkt Len Mean	Average packet size in the backward direction.
Packet-Level Features	Bwd Packet Length Std	Bwd Pkt Len Std	Standard deviation of packet sizes in the backward direction.
Flow Statistics	Flow Bytes/s	Flow Byts/s	Number of bytes per second for the flow.
Flow Statistics	Flow Packets/s	Flow Pkts/s	Number of packets per second for the flow.
Flow Statistics	Flow IAT Mean	Flow IAT Mean	Mean inter-arrival time (IAT) of packets in the flow.
Flow Statistics	Flow IAT Std	Flow IAT Std	Standard deviation of the inter-arrival time.
Flow Statistics	Flow IAT Max	Flow IAT Max	Maximum inter-arrival time between packets in the flow.
Flow Statistics	Flow IAT Min	Flow IAT Min	Minimum inter-arrival time between packets in the flow.
Forward (Fwd) and Backward (Bwd) Inter-Arrival Times	Fwd IAT Total	Fwd IAT Tot	Total inter-arrival time for forward packets.
Forward (Fwd) and Backward (Bwd) Inter-Arrival Times	Fwd IAT Mean	Fwd IAT Mean	Mean inter-arrival time between forward packets.
Forward (Fwd) and Backward (Bwd) Inter-Arrival Times	Fwd IAT Std	Fwd IAT Std	Standard deviation of forward inter-arrival time.

Feature Category	Feature Name (CICIDS2017)	Feature Name (CICFlowMeter)	Description
Forward (Fwd) and Backward (Bwd) Inter-Arrival Times	Fwd IAT Max	Fwd IAT Max	Maximum inter-arrival time between forward packets.
Forward (Fwd) and Backward (Bwd) Inter-Arrival Times	Fwd IAT Min	Fwd IAT Min	Minimum inter-arrival time between forward packets.
Forward (Fwd) and Backward (Bwd) Inter-Arrival Times	Bwd IAT Total	Bwd IAT Tot	Total inter-arrival time for backward packets.
Forward (Fwd) and Backward (Bwd) Inter-Arrival Times	Bwd IAT Mean	Bwd IAT Mean	Mean inter-arrival time between backward packets.
Forward (Fwd) and Backward (Bwd) Inter-Arrival Times	Bwd IAT Std	Bwd IAT Std	Standard deviation of backward inter-arrival time.
Forward (Fwd) and Backward (Bwd) Inter-Arrival Times	Bwd IAT Max	Bwd IAT Max	Maximum inter-arrival time between backward packets.
Forward (Fwd) and Backward (Bwd) Inter-Arrival Times	Bwd IAT Min	Bwd IAT Min	Minimum inter-arrival time between backward packets.
TCP Flag Features	Fwd PSH Flags	Fwd PSH Flags	Number of times the PSH flag was set in packets traveling in the forward direction.
TCP Flag Features	Bwd PSH Flags	Bwd PSH Flags	Number of times the PSH flag was set in packets traveling in the backward direction.
TCP Flag Features	Fwd URG Flags	Fwd URG Flags	Number of times the URG flag was set in packets traveling in the forward direction (dropped due to being constant).
TCP Flag Features	Bwd URG Flags	Bwd URG Flags	Number of times the URG flag was set in packets traveling in the backward direction (dropped due to being constant).

Feature Category	Feature Name (CICIDS2017)	Feature Name (CICFlowMeter)	Description
TCP Flag Features	Fwd Header Length	Fwd Header Len	Total header length of forward packets.
TCP Flag Features	Bwd Header Length	Bwd Header Len	Total header length of backward packets.
Additional Packet Count Features	Fwd Packets/s	Fwd Pkts/s	Number of forward packets per second.
Additional Packet Count Features	Bwd Packets/s	Bwd Pkts/s	Number of backward packets per second.
Flow Active and Idle Times	Min Packet Length	Pkt Len Min	Minimum packet length in the flow.
Flow Active and Idle Times	Max Packet Length	Pkt Len Max	Maximum packet length in the flow.
Flow Active and Idle Times	Packet Length Mean	Pkt Len Mean	Mean packet length in the flow.
Flow Active and Idle Times	Packet Length Std	Pkt Len Std	Standard deviation of packet lengths.
Flow Active and Idle Times	Packet Length Variance	Pkt Len Var	Variance in packet lengths.
TCP Flag Features	FIN Flag Count	FIN Flag Cnt	Number of packets with FIN flag.
TCP Flag Features	SYN Flag Count	SYN Flag Cnt	Number of packets with SYN flag.
TCP Flag Features	RST Flag Count	RST Flag Cnt	Number of packets with RST flag.
TCP Flag Features	PSH Flag Count	PSH Flag Cnt	Number of packets with PSH flag.
TCP Flag Features	ACK Flag Count	ACK Flag Cnt	Number of packets with ACK flag.
TCP Flag Features	URG Flag Count	URG Flag Cnt	Number of packets with URG flag.
TCP Flag Features	CWR Flag Count	CWE Flag Count	Number of packets with CWR flag.
TCP Flag Features	ECE Flag Count	ECE Flag Cnt	Number of packets with ECE flag.
Flow Statistics	Down/Up Ratio	Down/Up Ratio	Ratio of backward to forward traffic.
Flow Statistics	Average Packet Size	Pkt Size Avg	Average packet size in the flow.
Flow Statistics	Avg Fwd Segment Size	Fwd Seg Size Avg	Average segment size in the forward direction.
Flow Statistics	Avg Bwd Segment Size	Bwd Seg Size Avg	Average segment size in the backward direction.

Feature Category	Feature Name (CICIDS2017)	Feature Name (CICFlowMeter)	Description
Flow Active and Idle Times	Fwd Avg Bytes/Bulk	Fwd Byts/b Avg	Average number of bytes bulked in the forward direction.
Flow Active and Idle Times	Fwd Avg Packets/Bulk	Fwd Pkts/b Avg	Average number of packets bulked in the forward direction.
Flow Active and Idle Times	Fwd Avg Bulk Rate	Fwd Blk Rate Avg	Average bulk rate in the forward direction.
Flow Active and Idle Times	Bwd Avg Bytes/Bulk	Bwd Byts/b Avg	Average number of bytes bulked in the backward direction.
Flow Active and Idle Times	Bwd Avg Packets/Bulk	Bwd Pkts/b Avg	Average number of packets bulked in the backward direction.
Flow Active and Idle Times	Bwd Avg Bulk Rate	Bwd Blk Rate Avg	Average bulk rate in the backward direction.
Subflow Features	Subflow Fwd Packets	Subflow Fwd Pkts	Number of packets in the forward subflow.
Subflow Features	Subflow Fwd Bytes	Subflow Fwd Byts	Number of bytes in the forward subflow.
Subflow Features	Subflow Bwd Packets	Subflow Bwd Pkts	Number of packets in the backward subflow.
Subflow Features	Subflow Bwd Bytes	Subflow Bwd Byts	Number of bytes in the backward subflow.
Window and Segment Features	Init_Win_bytes_forward	Init Fwd Win Byts	Initial window size in bytes in the forward direction.
Window and Segment Features	Init_Win_bytes_backward	Init Bwd Win Byts	Initial window size in bytes in the backward direction.
Window and Segment Features	Fwd Act Data Pkts	act_data_pkt_fwd	Number of packets with actual data in the forward direction.
Window and Segment Features	Min Segment Size Forward	min_seg_size_forward	Minimum segment size observed in the forward direction.
Flow-based Timing Features	Active Mean	Active Mean	Mean time a flow was active before going idle.
Flow-based Timing Features	Active Std	Active Std	Standard deviation of time the flow was active.
Flow-based Timing Features	Active Max	Active Max	Maximum time a flow was active.
Flow-based Timing Features	Active Min	Active Min	Minimum time a flow was active.

Feature Category	Feature Name (CICIDS2017)	Feature Name (CICFlowMeter)	Description
Flow-based Timing Features	Idle Mean	Idle Mean	Mean time a flow was idle before becoming active again.
Flow-based Timing Features	Idle Std	Idle Std	Standard deviation of time the flow was idle.
Flow-based Timing Features	Idle Max	Idle Max	Maximum time a flow was idle.
Flow-based Timing Features	Idle Min	Idle Min	Minimum time a flow was idle.
Label	Label	Label	The class label indicating whether the flow is benign or belongs to a particular attack category

7. REFERENCES

[1]*Facts and Figures 2023 - Internet use*. (2023, October 10). https://www.itu.int/itu-d/reports/statistics/2023/10/10/ff23-internet-use/

[2]*IBISWorld - industry market research, reports, and statistics*. (n.d.). https://www.ibisworld.com/us/bed/internet-traffic-volume/88089/

[3] B.J. Radford, B.D. Richardson and S.E. Davis, "Sequence Aggregation Rules for Anomaly Detection in Computer Network Traffic," 2018. [Online]. Available: https://doi.org/10.48550/arxiv.1805.03735.

[4] *IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*. (n.d.). https://www.unb.ca/cic/datasets/ids-2017.html

[5] S. Garg et al., "A Hybrid Deep Learning-Based Model for Anomaly Detection in Cloud Datacenter Networks," *IEEE Transactions on Network and Service Management, vol. 16, no. 3, pp. 924-935, 2019. [Online].* Available: https://doi.org/10.1109/tnsm.2019.2927886.

[6]Siddiqui, M. A., Stokes, J. W., Seifert, C., Argyle, E., McCann, R., Neil, J., & Carroll, J. (2019). Detecting Cyber Attacks Using Anomaly Detection with Explanations and Expert Feedback. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. https://doi.org/10.1109/icassp.2019.8683212

[7]Nawir, M., Amir, A., Yaakob, N., & Lynn, O. B. (2019b). Effective and efficient network anomaly detection system using machine learning algorithm. *Bulletin of Electrical Engineering and Informatics*, *8*(1), 46–51. https://doi.org/10.11591/eei.v8i1.1387

[8] *The UNSW-NB15 Dataset | UNSW Research*. (n.d.). https://research.unsw.edu.au/projects/unsw-nb15-dataset

[9] Lin, P., Ye, K., & Xu, C. (2019). Dynamic network Anomaly Detection System by using deep learning techniques. In *Lecture notes in computer science* (pp. 161–176). https://doi.org/10.1007/978-3-030-23502-4_12

[10] *IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*. (n.d.). https://www.unb.ca/cic/datasets/ids-2018.html

[11] Hwang, R., Peng, M., Huang, C., Lin, P., & Nguyen, V. (2020). An unsupervised deep learning model for early network traffic anomaly detection. *IEEE Access*, *8*, 30387–30399. https://doi.org/10.1109/access.2020.2973023

[12] Wang, N. W., Zhu, N. M., Zeng, N. X., Ye, N. X., & Sheng, N. Y. (2017). Malware traffic classification using convolutional neural network for representation learning. *2017 International Conference on Information Networking (ICOIN)*. https://doi.org/10.1109/icoin.2017.7899588

[13] McDermott, C. D., Majdani, F., & Petrovski, A. V. (2018). Botnet Detection in the Internet of Things using Deep Learning Approaches. *International Joint Conference on Neural Networks 2018*. https://doi.org/10.1109/ijcnn.2018.8489489

[14] Lindemann, B., Maschler, B., Sahlab, N., & Weyrich, M. (2021). A survey on anomaly detection for technical systems using LSTM networks. *Computers in Industry*, *131*, 103498. https://doi.org/10.1016/j.compind.2021.103498

[15] Ullah, I., & Mahmoud, Q. H. (2021). Design and development of a Deep Learning-Based Model for Anomaly Detection in IoT networks. *IEEE Access*, *9*, 103906–103926. https://doi.org/10.1109/access.2021.3094024

[16] *The Bot-IoT Dataset | UNSW Research*. (n.d.). https://research.unsw.edu.au/projects/bot-iot-dataset

[17] Koroniotis, N., Moustafa, N., Sitnikova, E., & Turnbull, B. (2018, November 2). *Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: BoT-IoT Dataset*. arXiv.org. https://arxiv.org/abs/1811.00701

[18] Kim, H. K. (2019, September 27). *IoT network intrusion dataset*. IEEE DataPort. https://ieee-dataport.org/open-access/iot-network-intrusion-dataset

[19] Nassif, A. B., Talib, M. A., Nasir, Q., & Dakalbab, F. M. (2021). Machine Learning for Anomaly Detection: A Systematic Review. *IEEE Access*, *9*, 78658–78700. https://doi.org/10.1109/access.2021.3083060

[20] Sayed, M. S. E., Le-Khac, N., Azer, M. A., & Jurcut, A. D. (2022). A Flow-Based anomaly detection approach with feature selection method against DDOS attacks in SDNs. *IEEE Transactions on Cognitive Communications and Networking/IEEE Transactions on Cognitive Communications and Networking*, *8*(4), 1862–1880. https://doi.org/10.1109/tccn.2022.3186331

[21] Ullah, I., & Mahmoud, Q. H. (2022). An Anomaly Detection Model for IoT Networks based on Flow and Flag Features using a Feed-Forward Neural Network. *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. https://doi.org/10.1109/ccnc49033.2022.9700597

[22] Hephzipah, J., Vallem, R. R., Sheela, M., & Dhanalakshmi, G. (2023). An efficient cyber security system based on flow-based anomaly detection using Artificial neural network. *Mesopotamian Journal of Cybersecurity Vol. 2023*, 48–56. https://doi.org/10.58496/mjcs/2023/009

[23] Wang, Y., Houng, Y., Chen, H., & Tseng, S. (2023). Network Anomaly Intrusion Detection based on deep learning approach. *Sensors*, *23*(4), 2171. https://doi.org/10.3390/s23042171

[24] *KDD Cup 1999 Data*. UCI Machine Learning Repository. (n.d.). https://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data

[25] Stolfo, S., Fan, W., Lee, W., Prodromidis, A., & Chan, P. (1999). KDD Cup 1999 Data [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C51C7N.

[26] Ghulam Mohi-ud-din, December 29, 2018, "NSL-KDD", IEEE Dataport, doi: https://dx.doi.org/10.21227/425a-3e55.

[27] Song, J., Takakura, H., Okabe, Y., Et o, M., Inoue, D. & Nakao, K. 2011.Statistical Analysis of Honeypot Data and Building of Kyoto 2006+ Dataset for NIDS Evaluation. In: Proc. 1st Work-shop on Building Anal. Datasets and Gathering Experience Returns for Security. Salzburg, pp.29-36. April 10-13.

[28] Source, E. D. (n.d.). *IMPACT - Kyoto 2006+ Dataset*. https://www.impactcybertrust.org/dataset_view?idDataset=918

[29] K. Cho, K. Mitsuya, and A. Kato, “Traffic data repository at theWIDE project,” in USENIX 2000 Annual Technical Conference:FREENIX Track, June 2000, pp. 263–270.

[30] Mawi Working Group Traffic Archive. (n.d.). https://mawi.wide.ad.jp/mawi/

[31] *Caida*. CAIDA - The Center for Applied Internet Data Analysis. (2024, May 6). https://www.caida.org/

[32] Defcon.org. The Defense Readiness Condition - (n.d.). https://defcon.org/

[33] Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." Military Communications and Information Systems Conference (MilCIS), 2015. IEEE, 2015.

[34] Moustafa, Nour, and Jill Slay. "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 dataset and the comparison with the KDD99 dataset." Information Security Journal: A Global Perspective (2016): 1-14.

[35] Moustafa, Nour, et al. "Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks." IEEE Transactions on Big Data (2017).

[36] Moustafa, Nour, et al. "Big data analytics for intrusion detection system: statistical decision-making using finite dirichlet mixture models." Data Analytics and Decision Support for Cybersecurity. Springer, Cham, 2017. 127-156.

[37] Sarhan, Mohanad, Siamak Layeghy, Nour Moustafa, and Marius Portmann. NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems. In Big Data Technologies and Applications: 10th EAI International Conference, BDTA 2020, and 13th EAI International Conference on Wireless Internet, WiCON 2020, Virtual Event, December 11, 2020, Proceedings (p. 117). Springer Nature.