THE CATHOLIC UNIVERSITY OF AMERICA

Anomaly Based Intrusion Detection System in Network

A THESIS

Submitted to the Faculty of the

Department of Electrical Engineering and Computer Science
School of Engineering
Of The Catholic University of America
In Partial Fulfillment of the Requirements
For the Degree
Master of Science

By

Nafiz Mahamud
Washington, D.C.

2024

Anomaly Based Intrusion Detection System in Network

Nafiz Mahamud, M.S.

Director: Minhee Jun, Ph.D.

Abstract

Due to the rapid growth in network traffic and increasing security threats, Intrusion Detection Systems (IDS) have become increasingly critical in the field of cyber security for providing secure communications against cyber adversaries. We are going to detect intrusions in a network based on anomalies. We have applied supervised machine learning algorithms like the random forest, and support vector machines on datasets such as NSL-KDD and CICIDS2017.

Moreover, we will focus on semi-supervised deep learning on a practical dataset, called Iot-23. Since intrusions are a series of related malicious actions performed by an internal or external intruder that attempts to compromise the targeted system, we are interested in applying sequence learning and already have applied Recurrent Neural Network(RNN), Long-Short Term Memory(LSTM), and Gated Recurrent Units(GRU) and got pretty promising results so far. The simulation results show that the proposed models and methods on the dataset are efficient in time as well since we select the most relevant features to reduce the dimensionalities. In other words, the model will detect an intrusion within a very short time. Therefore, our model can be used to detect an intrusion like in a vehicle network, power consumption, etc.

This thesis by Nafiz Mahamud fulfills the thesis requirement for the Master of science degree in approved by Minhee Jun, Ph.D. as Director, and by Chaofan Sun, Ph.D. as Reader.

Minhee Jun

Minhee Jun, Ph.D., Director

Chaofan Sun, Ph.D., Reader

Contents

1. Introduction

1.1 Motivation and Context

Telecommunication technologies are evolving rapidly, making the world more connected each year. New standards and technologies, such as 5G - the fifth-generation standard for cellular networks - and Internet-of-Things (IoT) devices, are getting widely adopted in modern industries, including automotive transport applications, smart agricultural farming, and public safety services [1]. The increasing demand for connectivity necessitates raising the standards for security. With more at stake, network security solutions of tomorrow will have to become more automated, scalable, and reliable [2]. There exist many different approaches to network security; for instance, firewalls, email security, and anti-malware software are some of the standard methods for protecting personal computers.

In this work, we are interested in the approach called network intrusion detection. The purpose of a network intrusion detection system (NIDS) is to monitor the network's activity and detect in real time various harmful events, such as misconfigurations of network devices or cyber-attacks. A popular approach, also adopted in this work, is to apply anomaly detection techniques to network intrusion detection [3, 4]. An anomaly-based network intrusion detection system (ANIDS) detects malicious network activities by searching for abnormal patterns in network traffic. Figure 1 illustrates the typical flow of an ANIDS. Such systems have many appealing properties, such as detecting novel zero-day attacks. However, certain aspects make a successful deployment of an ANIDS in the real-world environment difficult. According to Sommer et al. [5], the challenges include "a high cost of errors, lack of labeled data, high complexity and variability in input data, and fundamental difficulties for evaluating the system." To overcome

these challenges, we need to understand better both the system under protection, as well as the capabilities and limitations of the detection process.

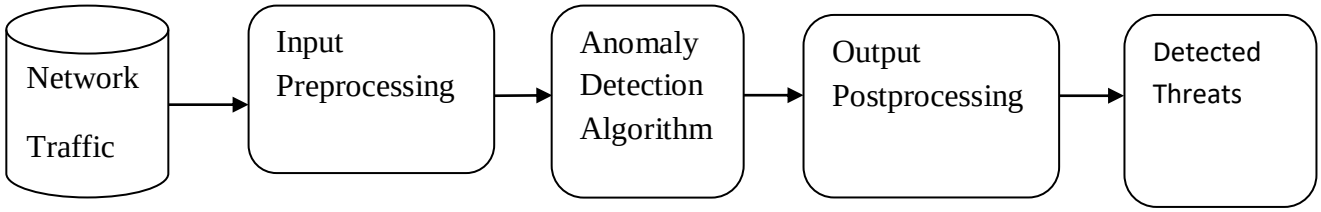

Figure 1: Typical flow of anomaly-based network intrusion detection system [1]

Like in any machine learning (ML) application, data is an essential part of the problem. A network intrusion dataset, containing traffic data collected from some network (typically with traces of executed attacks), is necessary for the development and evaluation of anomaly-based network intrusion detection techniques. Due to the complex nature of computer networks, each network intrusion dataset has its own specific characteristics and idiosyncrasies. For example, one dataset may contain traffic from a university network, and another from an industrial IoT network. The statistical nature of the traffic is very different in both cases, as well as the relevant cyber-threats. For this reason, researchers commonly agree that it is crucial to use several datasets to make any general conclusions in this domain [5, 6]. Fortunately, in the few recent years, a number of representative network intrusion datasets have been created and made

publicly available [7, 8, 9, 10]. The availability of modern network intrusion datasets opens new opportunities for research. At the same time, the research on anomaly detection methods has seen significant advancements. Recently, deep learning (DL) techniques for anomaly detection have become increasingly popular among researchers and practitioners [11]. In particular, recurrent neural networks and auto-encoding methods based on artificial neural networks (ANNs) have shown state of- the-art results on the benchmark anomaly detection datasets [12, 13, 14]. Deep learning methods seem to be particularly appealing for network intrusion detection, as they are able to extract rich representations from input and scale well to large datasets [15, 11]. To the best of our knowledge, comprehensive algorithmic studies in the field of network intrusion detection are few and far between. Many papers proposing new methods use only one dataset for evaluation or rely on outdated datasets, some of which have been shown to be unrealistic, such as KDDCUP'99 [16]. Consequently, we observe the potential for more thorough studies that evaluate state-of-the-art anomaly detection techniques on latest representative network intrusion dataset called "Iot-23". Multiple trained Deep Learning models are tested on this dataset. The reason for choosing multiple models is to fit the individual needs for different users or groups. In other words, it is important to find the efficient model for different type of user.

1.2 Purpose

Given the context presented in Section 1.1, this work provides a comparison of anomaly detection algorithms and techniques for network intrusion detection. Specifically, we compare Recurrent Neural Networks-based models operating in the semi-supervised mode against classical anomaly detection methods. Furthermore, we analyze the difference in performance of the models trained on the network flow and aggregated-flow features. To the best our knowledge, such a study has not been conducted yet. Our primary aim is to study the baseline detection performance that can be achieved using the selected techniques. Additionally, we

pursue gaining useful insight on the semantics of the detection process.

1.3 Structure of the Thesis

The remainder of this thesis is organized as follows:

• Section 2 reviews the previous work related to the objective of applying anomaly detection methods to network intrusion detection.

• Section 3 provides background information on network security and anomaly Detection and also an overview of the related algorithms.

• Section 4 gives an overview of the selected network intrusion dataset, identifying their main properties and describes the selected methodologies.

• Section 5 describes our methods and comparison methodology, and gathers the obtained results.

• Section 6 identifies limitations of and future directions for this work.

2 Related Work

Anomaly-based network intrusion detection is a large and actively developing research area. In what follows, we review some of the influential past publications that are related to our work. Several non-parametric statistical-based methods have been proposed for network intrusion detection. Kruegel and Vigna [18] present an intrusion detection system to detect attacks against web servers and web-based applications. The system takes as input web server logs and derives automatically statistical profiles associated with the referenced web application. The learned profiles are then used to assign an anomaly score for each web request. Mahoney and Chan [19] propose two probabilistic models: the packet header anomaly detection (PHAD) and the application layer anomaly detection (ALAD) model. The models use non-stationary statistical profiling, in which probabilities are estimated based on the time since the last observation of an event rather than its average rate. The PHAD model inspects network packets using the header attributes. The ALAD model inspects server TCP connections using such attributes as the application protocol keywords, TCP flags, IP addresses, and port numbers. The models are evaluated on the 1999 DARPA IDS dataset (cf. Section 5). Parametric statistical techniques have also been used for modeling network traffic. Simmross-Wattenberg et al. [20] present a two-stage approach for detecting network traffic anomalies. First, the authors use α-stable first-order distributions to model 30-minute windows of network traffic. Second, they apply generalized likelihood ratio test to classify whether the traffic windows are anomalous. The authors focus on detecting two types of attacks, namely floods and flash crowds, and evaluate their method on a custom dataset containing traffic from two university routers. Nguyen et al. [21] propose using the Holt-Winters algorithm, a classic forecasting technique for time series data, to find anomalies in network traffic. The authors apply Holt-Winters to monitor four metrics extracted

from the network flow data. Anomalies are detected when any of the metric observations falls outside of the predicted range. For evaluation, the authors create a custom dataset that contains traces of the flooding and port scanning attacks.

Clustering and k-nearest neighbor (kNN) techniques have been proposed for network intrusion detection. Eskin et al. [22] present a framework for unsupervised anomaly detection. First, they apply two mappings to transform the input feature space: a data normalization mapping and a spectrum kernel mapping. Then, they train three algorithms on a set of unlabeled data points from the transformed space.

Specifically, they compare a clustering-based model, a kNN model, and a support vector machine (SVM) model. The clustering algorithm produces fixed-width clusters in one pass through the data; the data points in the small clusters are detected as anomalies. For evaluation, the authors use the KDDCUP'99 dataset. García et al. [7] propose a clustering-based method called BClus. They use the Expectation- Maximization algorithm to cluster the aggregated network flow data. In the second stage, they use RIPPER, a propositional rule learning algorithms, to classify the obtained clusters as normal or anomalous. The authors evaluate BClus on the CTU-13 botnet dataset (presented in the same paper).

Several publications have studied the effectiveness of principal component analysis (PCA) for network intrusion detection. Ringberg et al. [23] identify the main challenges of using PCA to detect traffic anomalies. They show that tuning PCA to effectively operate in a real-world environment is difficult and requires more robust approaches than a straightforward one. Brauckhoff et al. [24] show that simple PCA fails to capture the temporal correlation of network traffic and propose replacing it with the Karhunen-Loeve expansion. Kanda et al. [25] present a PCA-based algorithm called ADMIRE for detecting anomalies in flow-based data. ADMIRE uses three-step random projections and an adaptive parameter setting to improve detection performance. The authors use the MAWI dataset for evaluation and conclude that ADMIRE

outperforms a classical PCA-based detector. Numerous classification-based methods for network intrusion detection have been proposed. Moustafa and Slay [26] compare a selection of classification algorithms, including naïve Bayes, decision trees, ANNs, and logistic regression on two datasets: the UNSW-NB15 dataset (presented in the same paper) and KDDCUP'99. Bilge et al. [27] propose a flow-based botnet detection system called Disclosure. The authors train (in a supervised way) a random forest classifier to distinguish command and control (C&C) communication from benign network traffic. To reduce the false positive rate, Disclosure uses external reputation scores, such as Google Safe Browsing. For evaluation, the authors use two network environments: a medium-size university network and a tier 1 internet server provider (ISP) network. Recently, deep learning-based anomaly detection methods have become popular among researchers [11]. Kwon et al. [28] apply various deep learning models, including multilayer perceptrons (MLPs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs) for anomaly detection in network traffic. The authors use three public datasets for evaluation: NSL-KDD, Kyoto Honeypot, and MAWI. Torres et al. [29] propose using a large short term memory (LSTM) network to detect malicious behavior by modeling network traffic as a sequence of states that change over time. The authors evaluate the model on two captures from a university network that contain traces of malicious botnet activities. They show that LSTM achieves good performance overall but struggles with a few cases, where the botnet behavior is not as easily differentiable.

Numerous recent works propose using deep learning-based autoencoding (AE) models for network intrusion detection. Several authors [30, 31, 32] adopt a twostage approach: first, they use autoencoding-based models to extract compact representations of data points; second, they use the learned representations as input features for standard supervised classifiers, such as logistic regression. A more interesting and practical approach is to train autoencoding models in the fully unsupervised or semi-supervised mode. Mirsky et al. [33] present Kitsune - a novel

autoencoder-based NIDS, which can learn to detect network attacks in an online and unsupervised manner. The authors mainly focus on the scalability of the system; however, they claim Kitsune achieves performance comparable to offline anomaly detectors. The evaluation is performed on a custom dataset. Further, multiple publications [34, 35, 17, 36] propose using more advanced probabilistic versions of autoencoders, such as the variational autoencoder (VAE) and the conditional variational autoencoder (CVAE). An and Cho [34] propose using the reconstruction probability as the anomaly score for VAE. They show that VAE outperforms AE- and PCA-based methods on the benchmark datasets, including KDDCUP'99. Xu et al. [35] apply VAE for detecting anomalies in seasonal key performance indicators of web applications. Although this security task is different from network intrusion detection, the authors show that their unsupervised method outperforms a supervised ensemble approach by a wide margin. Despite a surging number of publications proposing novel methods for network intrusion detection, to the best of our knowledge, there seems to be a lack of comprehensive studies, that undertake a rigorous comparison of methods and evaluate them over several representative datasets. A work that gets closer to the topic in hand is that of Falcão et al. [37], which compares a total of 12 unsupervised anomaly detection (statistical, neighbour-based, density-based, and classificationbased) algorithms on five network intrusion datasets (KDDCUP'99, NSL-KDD, ISCX2012, ADFA-LD, and UNSW-NB-15). The authors identify the families of algorithms that are more effective for network intrusion detection and the families that are more robust to the choice of configuration parameters. Another study by Skvára et al. [12] compares deep generative autoencoding models (including VAE) against classical algorithms on a large number of benchmark datasets in semi-supervised and unsupervised settings. The authors conclude that the performance of the generative models is determined by the selection process of their hyperparameters. However, the study does not consider the domain of network intrusion detection. Several previous works [7, 17] propose aggregating network flows; however, to the

best our knowledge, no research on the efficacy of this method has been published. Our work presents a thorough comparison of deep autoencoding-based models and classical methods for network intrusion detection. We train the models in a semi-supervised fashion and evaluate them on the latest representative modern dataset.

According to [13], a mechanism with low computational complexity has been proposed by using, random hopping sequence and random permutations to hide valuable information. Moreover, in [14], Doshi presented a method to detect DDoS attacks in the network layer with low-cost machine learning approach, including KNN, LSVM, NN, Decision Tree, and Random Forest. This method can detect which node is attacking the central unit with IP address. This method was reported to achieve high testing accuracy for all five machine learning algorithms. In [21] detection of anomaly is done using the fog computing, which clusters the different types of anomalies present in the sensor layer or edge nodes without performing computation on both the cloud and sensor layer but in the fog layer of the network. By using the fog computing method it has become more easy to detect an anomaly. In [17], the author tries to implement malware detection system by using different classifiers of k-NN and random forest to build the model. The device filters TCP packets and selects important features such as frame numbers, length, labels etc. The k-NN algorithm assigns traffic to the class while the random forest classifier builds decision trees to detect the malware. The authors have proposed a new methodology in [22] which uses game theory and nash equilibrium to help the resource constrained IoT devices to detect an anomaly using Intrusion Detection System(IDS), activating it only when needed. When an attack occurs the attack pattern (signature) is stored and then model is trained and whenever pattern repeats it is identified by the signature detection technique and anomaly is detected. Using IDS all the time can be resource consuming, so the game theory and nash equilibrium come into place to determine when to activate the IDS to detect an anomaly and to add a new rule to signature pattern and build the model. Machine

learning or Deep Learning methods have been discussed in [16]. The various types of attacks at different levels of IoT infrastructure are clearly explained and the possible solutions to these attacks using Machine learning are also clearly explained, that which are caused due to the lack of proper security data available, the low quality data available and performance of the learning algorithms could be the key in providing and improving the security and privacy of IoT devices. In this paper we would like to calculate the accuracy for the models, thereby comparing them to get the model that gives highest accuracy with less false negative rate to detect and prevent the malware attacks in resource constrained IoT devices.

3 Background

In this section, we describe our main two separated fields to understand our project: network security and intrusion detection and also give an overview of the related algorithms.

3.1 Network Security

In this subsection, we first explain the basic concepts of networking and principles of communication. We discuss the TCP/IP model and the principles of network communication protocols, such as IP, TCP and UDP. Finally, we look at the different phases of a cyber-attack and give an overview of the cyber-threat landscape.

3.1.1 Networking Fundamentals

Communication is a complex process. When a group of people plans to engage in a conversation, they first need to find a place where they can hear each other and agree on the language they will speak. They will likely start with greetings and introductions, following appropriate social norms. The process is similar in computer networks, albeit more complicated and less arbitrary. A computer network is a group of connected devices that can communicate with each other. Network devices, also called network nodes, can be personal computers, servers, phones, IoT devices, and others. Network nodes are interconnected utilizing some communication media that can be either cable (e.g., twisted pair or optical fiber) or wireless (e.g., WiFi, cellular). Computer networks are generally classified into [38]:

a. Local Area Networks (LAN): A LAN is a small private network, usually limitedto a single building.

b. Metropolian Area Network (MAN): A MAN is a larger kind of network that typically covers a whole city/metropolitan area.

c. Wide Area Network (WAN): A WAN is the largest kind of networks that typically covers whole countries/continents

For historical reasons, there exist two reference models that describe how communication systems operate: the Open Systems Interconnection (OSI) model and the Internet protocol suite, also known as TCP/IP. The former is a more comprehensive reference framework for general networking systems, while the latter is used in the Internet and similar computer networks . The TCP/IP model is sometimes seen as a concise version of the OSI model [39], for which reason we will use it in this thesis. The TCP/IP model follows the layering principle, dividing a communication system into four abstraction layers, as described in Table 1. Each layer provides some specific functionality to the layer above it and uses the services of the layer below it. For example, the transport layer relies on the internet layer and serves the application layer. Interactions between network devices at the save layer follow a system of rules called a communication protocol. Communication protocols are designed to exchange specific units of data called protocol data units (PDU). The layer at the destination receives the PDU sent by the layer at the source. For example, the PDU of the internet layer is a data packet.

In what follows, we will describe the main layers and protocols of the TCP/IP model:

a) Link layer. The link layer is responsible for the physical transmission of data within the local network segment (link) that a device is connected to. The protocols residing in this layer operate this layer operate at the hardware level.

b) Internet layer. The internet layer is responsible for exchanging data between networks. The principal protocol of this layer is the Internet protocol (IP). IP delivers packets of data from the source node to the destination node based on their IP addresses, which are unique numerical

labels that are assigned to all nodes. This process is called routing.

Layer	PDU	Function
Link	Frame	Moves packets between different hosts on the same local network link
Internet	Packet	Exchanges packets across network boundaries
Transport	Segment, Datagram	Maintains end-to-end communications across the network
Application	Data	Provides software applications with standardized data exchange.

Table 1: Layer architecture of the TCP/IP model, adapted from [39]

There are two versions of the Internet protocol: IPv4 and IPv6. The main difference is that IPv4 uses 32 bit IP addresses, and IPv6 - 128 bit IP addresses. As previously mentioned, data travels across the network in packets due to the physical limit on the amount of data that can be transmitted at one time. An IP packet consists of a header section that contains the source and destination IP addresses, and a data section also called a payload. Figure 2 illustrates the structure of an IPv4 header. IP is a connectionless protocol, meaning that no session information is retained by either the source or the destination node. It is agnostic to the data structures at the transport level.

c) Transport layer. The transport layer is responsible for end-to-end communication between network nodes. In other words, it ensures that data is transferred from the source node to the destination node, offering control over the reliability of the transmission. The two main protocols residing in the transport layer are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Both protocols segment the data into pieces that get encapsulated in

IP packets and sent across the network using the IP protocol.

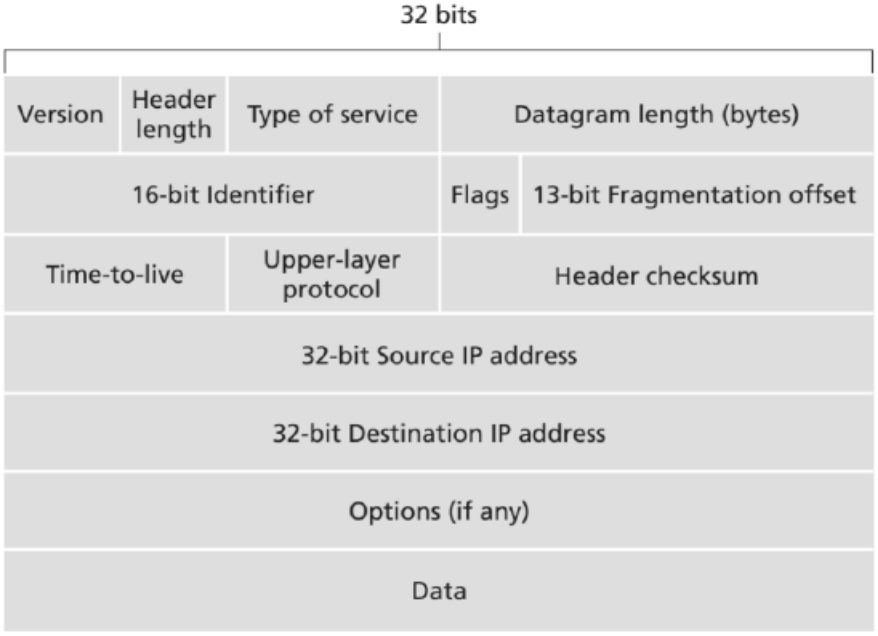

Figure 2: The IPv4 header, taken from Computer networking : a top-down approach[38]

TCP provides reliable transmission of data, ensuring that all the packets arrive in order and that there are no duplicated, corrupted, or lost packets. TCP is a connection-oriented protocol, meaning that it establishes a connection between the source node and the destination node before any data is sent and closes it afterward. TCP implements these control mechanisms using special bits in its header called TCP flags; e.g., the ACK flag that is used for acknowledgment. Figure 3 illustrates the structure of a TCP header. TCP is a complex protocol and has high latency due to its reliability features. UDP is a connectionless transport protocol that provides "unreliable" transmission of data. Unlike TCP, it does not establish a connection between nodes and does not feature error- control mechanisms. UDP is useful in applications like audio or video streaming where low latency is more important than reliability.

Figure 4 illustrates the structure of a UDP header. Transport protocols use network ports to allow multiple conversations between network nodes simultaneously. A network port is a logical

construct (represented using an unsigned number) that identifies a specific process or service on a host, which is itself identified by the IP address. Network ports are specified in the transport protocol headers, while IP addresses are specified in the IP headers.

d) Application layer.

The application layer provides software applications with data exchange services established by the lower layers. Examples of protocols residing in the application layer are:

– Hypertext Transfer Protocol (HTTP) is extensively used in the World

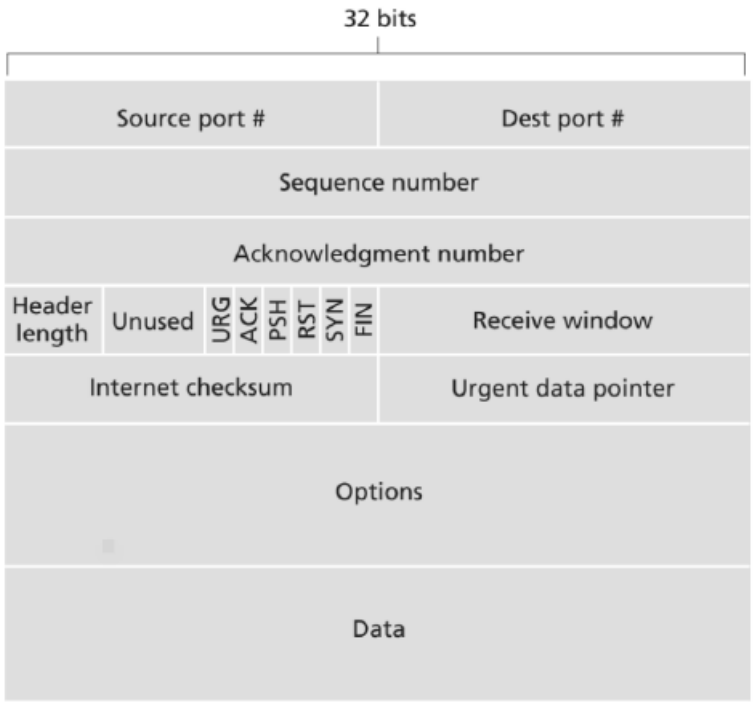

Figure 3: The TCP header, taken from [38]

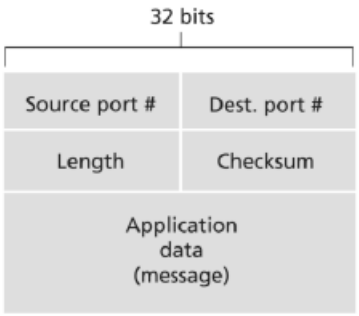

Figure 4: The UDP header, taken from [38]

Wide Web for managing data communication between web clients and servers. HTTPS is a secure version of HTTP.

– *File Transfer Protocol (FTP)* is used for transferring files between a client and server.

– *Simple Mail Transfer Protocol (SMTP)* is used for electronic mail exchange.

– *Internet Relay Chat Protocol (IRC)* is used for communication in the form of text.

– *Message Queuing Telemetry Transport Protocol (MQTT)* is a lightweight messaging protocol for IoT devices, optimized for high latency or unreliable networks.

3.1.2 Cyber Threat Landscape

In this section, we describe common types of cyber-attacks and give a brief overview of the cyber threat landscape.

A cyber-attack is any kind of malicious activity that targets computer networks or devices [3]. Attackers try to exploit network device or protocol vulnerabilities to gain unauthorized access, steal data, or disrupt the normal functioning of the system. Their motives can vary from personal

gain to military intelligence. A kill chain is a military concept that describes the structure of an attack. Although not all cyber attacks follow every phase of the kill chain, it is a useful tool for attack analysis and defense planning. Table 2 explains the kill chain phases.

Phase	Actions
Reconnaissance	Research, identification and selection of targets
Weaponization	Pairing remote access malware with exploit into a deliverable payload
Delivery	Transmission of weapon to target
Exploitation	Once delivered, the weapon's code is triggered exploiting vulnerable applications or systems
Installation	The weapon installs a backdoor on a target's system, allowing persistent access
Command & Control	Outside server communicates with the weapons
Actions on Objective	The attacker works to achieve the objective of the Intrusion

Table 2: Phases of the intrusion kill chain, adapted from [40]

Some common types of cyber-attacks include:

a. Network scanning. In a network scanning attack, the adversary attempts to discover listening ports on the devices in a network. To achieve it, the adversary selects a list of port numbers and tries to establish a connection on each port. More specifically, there are three kinds of port scanning attacks:

– Vertical scan: a scan against a group of IPs for a single port

– Horizontal scan: a scan against a single IP for a group of ports.

– Box scan: a combination of the above.

The purpose of network scanning is reconnaissance: the adversary's goal is to gather information on the target infrastructure, which will be utilized in the next phase of the attack.

b. Denial-of-Service (DoS) attacks. In a DoS attack, the adversary attempts to disrupt the normal functionality of a network service, typically to make it unavailable for the users. DoS attacks can target a single computer, e.g., a bank server, or an entire network. DoS attacks are typically executed by flooding the target with traffic, exhausting its capacity to process it. When a DoS attack is carried out by multiple computers, it is called a Distributed Denial-of-Service attack (DDoS). Naturally, the more resources are used in the attack, the more powerful and disruptive it is.

There are three main categories of DDoS attacks:

– Application layer attacks. This category of DDoS attacks target the application layer, where common web activities take place. For example, in an HTTP flood attack, the adversary uses his computational resources to overwhelm the target server with HTTP requests, until it is unable to process normal traffic, as depicted in Figure 6. Application layer attacks are particularly effective because they deplete the server as well as the network resources. Furthermore, they can be hard to detect in time if the malicious requests resemble normal user activity [41].

– Protocol layer attacks. This category of DDoS attacks exploit weaknesses in the transport and internet protocols. For example, in a SYN Flood attack, the adversary misuses the TCP protocol by sending a large number of connection requests to the target at once, never finalizing the connection processes. Other examples of protocol-level DDoS attacks include ACK Flood, SYN-ACK Flood, and UDP flood.

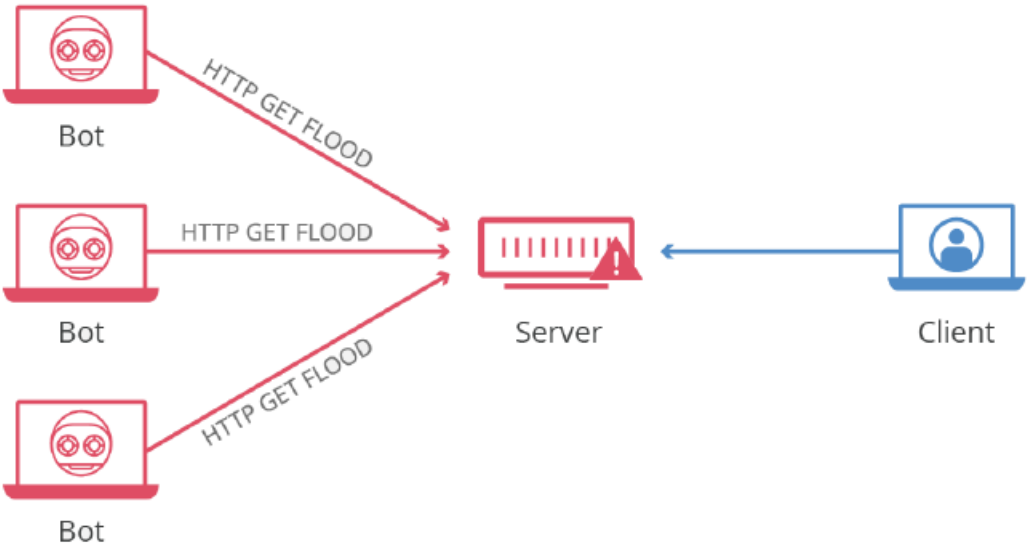

Figure 5: HTTP Flood attack, taken from [41]

– Volumetric attacks. This category of DDoS attacks attempts to consume all the network resources of the target by sending a massive amount of traffic to the target. Often the adversary employs some technique to amplify the amount of data that is sent. For example, in a DNS amplification attack, the attacker sends fake requests to open DNS servers, making them send large replies to the target.

c. Botnets. A botnet is a group of network devices, each of which has been compromised by an adversary, falling under his control. In this case, each compromised device is called a bot, and the adversary is called the botmaster. The botmaster can use various methods to create bots, like guessing or stealing authentication credentials or installing a piece of malware. The power of botnets lies in their ability to self-propagate; in other words, the existing bots can "recruit" new bots in their surrounding network. With an efficient recruitment method, the attacker can grow the botnet at an exponential rate. Once large enough, botnets can be used to carry out various malicious activities, such as DDoS attacks, spam distribution, and cryptomining.

To launch an attack, the attacker needs to be able to send commands to the bots. The communication happens through standard network protocols, such as HTTP or IRC. There are two main control structures:

– Client/server. This is a centralized control structure where the bots receive commands from the Command and Control (C&C) servers (Figure 6). The advantage of this architecture (from the attacker's point of view) is its simplicity: operating the botnet is straightforward and fast. The disadvantage is that it has a single point of failure - if the defenders manage to identify and bring down the C&C servers, the whole botnet is disrupted.

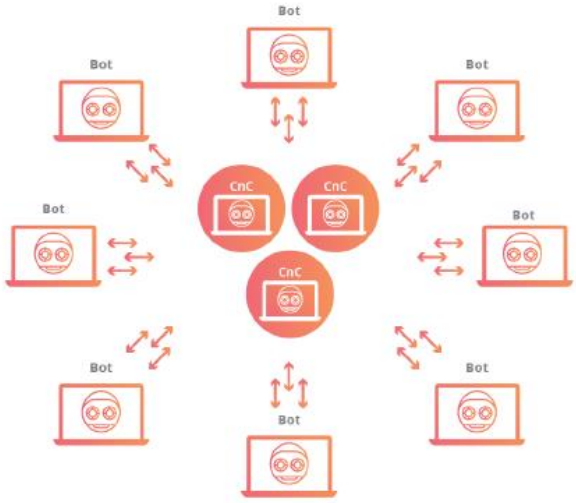

Figure 6: Client/server botnet architecture, taken from [41]

– Peer-to-peer (P2P). This is a decentralized control structure - bots take the role of control centers, sending commands to their peers and receiving commands themselves, as illustrated in Figure 7. The advantage of this architecture is that it does not have a single point of failure: bringing down one bot does not disrupt the whole botnet. The price to pay is the increased latency and difficulty of controlling the botnet.

In 2016, one of the largest DDoS attacks in history was carried out using an IoT botnet that was later named "Mirai" [42]. IoT devices often have serious security vulnerabilities that makes them

perfect bot candidates. The Mirai creators used a very simple, yet astonishingly successful technique to recruit new bots; in essence, they were trying to login into devices using a predetermined list of username and password pairs. The Mirai threat has not been stopped; additionally, more sophisticated versions of the botnet, such as “Okiru”, have emerged since the first attack. ENISA Threat Landscape Report 2018 [43] ranks IoT botnets among the top modern cyber threats.

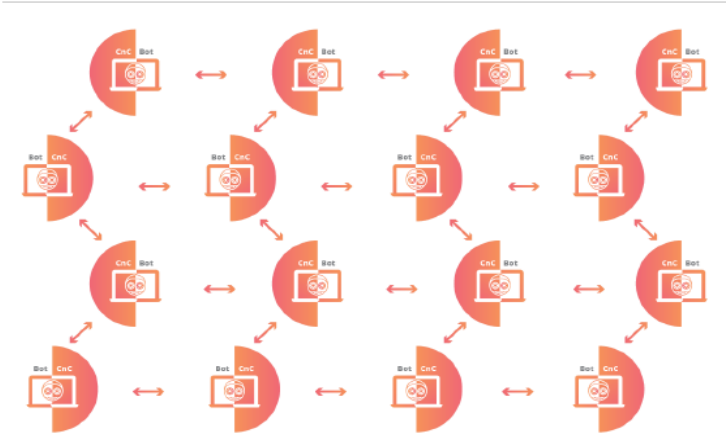

Figure 7: Peer-to-peer botnet architecture, taken from [41]

3.2 Intrusion Detection System in Network

There are mainly two intrusion detection system (IDS): one is signature based intrusion detection and another is anomaly based intrusion detection system. We are going to give a short preview about those systems.

3.2.1 Signature based Intrusion Detection System (SIDS)

Signature-based intrusion detection system (SIDS) is a method used in cyber-security to identify and prevent known threats by comparing network or system activity against a database of predefined signatures. These signatures are essentially patterns or characteristics that are associated with known types of attacks, malware, or malicious activities. The goal is to detect and block malicious behavior by recognizing patterns that match those of previously identified

attacks. Although it can get pretty much high accuracy, now-a-days SIDS is no longer a hot research topic due to inability to detect new threats and new attacks.

Because of the advancement of technologies, the number of attacks are increasing exponentially. Statistics show that attacks number increases with a rate of 100% each year causing huge money loss, about tens of millions of dollars for ran-somware attacks only [41]. This high number of millions of new threats that are developed every day, reduces the effectiveness of signature-based IDS because it is not a practical solution to update the signatures databases every few minutes. Therefore, Anomaly-based IDS can be a better alternative of signature-based IDS because it can detect new threats.

3.2.2 Anomaly based Intrusion Detection System

In this section, we are going to discuss what is anomaly based intrusion detection system (AIDS), why it is importance and what are the challenges. We will give a brief description about the existing techniques. After that, we will finally give the background of applied algorithms.

What is anomaly?

In the context of Anomaly based Intrusion Detection System (AIDS), an anomaly refers to an observation or event that deviates significantly from the expected or normal behavior within a given dataset. But, an anomaly can be an intruder/threat or it cannot be the threat. Therefore, we have to build a model that can understand the context of the relevant features to detect the anomaly. We discuss it in the following section.

In practice, anomaly detection is used to solve detection problems of some kind, usually concerning defects, fraud, or other undesirable events. Example applications include credit card fraud, medical disease diagnosis, natural disaster prediction, and, last but not least, network intrusion detection. The main aspects of an anomaly detection problem are the nature of input data, the type of anomalies, and the availability of labels. All of them have to be taken into

account when choosing a suitable anomaly detection technique.

We usually see in a classic machine learning task that data can be in the form of a table, image, text, or sequence. We often feel the necessity to perform feature engineering, which might require domain expert knowledge. The nature of data might limit the choice of a suitable technique; for example, text mining methods are unlikely to be directly applicable to image data. In this thesis, we are dealing with tabular data that has mixed categorical and numeric features. The data points and their attributes are further discussed in Sections 4.

Anomalies are typically classified into three categories[17]

a) Point anomaly. A point anomaly is a single data point that is found anomalous with respect to the rest of the data. For example, a thief makes an expensive purchase with a stolen card from an otherwise low-expenditure bank account.

b) Contextual anomaly. A contextual anomaly is a single data point that is found anomalous in a certain context. For instance, a thief uses a stolen card to make a regular transaction at an usual time or geographical location.

c) Collective anomaly. A collective anomaly is a set of data points that is found anomalous with the rest of the data. For example, a thief carries out a series of otherwise regular transactions on an irregular basis, forming a suspicious pattern.

Anomaly detection techniques vary and may include statistical methods, machine learning algorithms, or a combination of both. These methods learn from historical data to establish a model of normal behavior and identify instances that do not conform to this model. For example,

techniques for detecting point, contextual, and collective anomalies are different, with the majority of existing methods aiming at point anomaly detection, as this notion of anomaly is the simplest. Techniques for detecting contextual or collective anomalies are highly domain-specific. However, it is sometimes useful to reduce contextual or collective anomalies to point anomalies. Obtaining labels for an anomaly detection task is typically hard due to the nature of the problem; indeed, in domains like earthquake detection, anomalies are scarce and limited data is available. Furthermore, abnormal behavior can be highly complex and change over time, in which case collecting and labeling enough data to capture it might not be feasible. On the other hand, data describing normal behavior is usually abundant.

Depending on the label availability, anomaly detection algorithms can operate in three modes:

a) Supervised mode. During training, an algorithm has access to both normal and anomalous data points and their corresponding labels. In the stolen-card example from above, the training data contains both normal and fraudulent transactions with the corresponding labels.

b) Semi-supervised mode. During training, an algorithm has access to normal data points only. In the above example, the training data contains only normal transactions (only one label is present).

c) Unsupervised mode. During training, an algorithm experiences data without any labels. In the above example, the training data contains normal and possibly fraudulent transactions (no labels are present).

Unsupervised anomaly detection methods typically require making strong assumptions about the data and perform efficiently only when those assumptions hold [22]. At the other end of the spectrum, supervised techniques are guided by the labels during training, similar to classic predictive models, and do not require making strong assumptions. Semi-supervised techniques are the middle ground between the two and are commonly viewed as being more applicable in practice [44, 12]. Semi-supervised models train on (mostly) normal data and can utilize a small number of anomalies for hyperparameter tuning. We adopt the semi-supervised approach in our experiments. No matter which operational mode is used, a labeled test set is required for evaluating the performance of the method in turn.

Anomaly detection algorithms typically assign an anomaly score to each input instance, which is "a measure of the degree to which that instance is considered an anomaly" [44]. In this work, we follow the convention that "the higher the score, the more likely it is that the instance is an anomaly." To flag all anomalous instances, we need to turn anomaly scores into predictions. A simple way to achieve that is to use a detection threshold. All the instances with scores higher than the threshold get flagged as anomalies. The threshold can be selected by analyzing the top few anomalies during the validation stage.

An anomaly-based NIDS (ANIDS) attempts to detect attacks by searching for unusual patterns in network traffic. The ANIDS employs some anomaly detection technique to learn a profile of the benign network traffic data. Any significant deviations from the learned benign profile are flagged as potential malicious activities.

The main assumption of this approach is that the attacks launched against the network will leave a trace in the network traffic that does not conform to the network's normal behavior. The ANIDS should ideally be able to detect known as well as unknown attacks without any prior knowledge because it does not rely on signatures [3] .

However, some challenges arise in practice [17]:

– Some network devices, such as servers or personal computers, exhibit a very complex and variant behavior. In such environments, it is unrealistic to expect anomaly detection techniques to perform efficiently. If the traffic is highly non-homogeneous, the ANIDS will produce a large number of false alarms (report benign traffic as suspicious), which may limit its usability [5].

– Network traffic may have periodic patterns or drift over time. For example, the amount of traffic may peak during the daytime and fall during the nighttime.

– Specific malicious activities, like C&C communication, can appear to be very similar to the benign traffic, often not without the attacker's efforts.

Table 3 summarises the advantages and disadvantages of signature-based and anomaly-based NIDS. A successful strategy for deploying a NIDS in a real-world environment may include both anomaly-based and signature-based approaches.

Property	Network Intrusion Detection System (NIDS)	
	AIDS	SIDS
Interpretability	+	-
Robustness	+	-
Maintenance	-	+
Unknown attack detection	-	+

Table 3: Advantages and disadvantages of using signature- and anomaly-based NIDS

Since we are focusing on the anomaly based intrusion dectection system, let's discuss a few more important points.

It should be mentioned that anomalous patterns in network traffic may not necessarily be caused

by attacks and exploits. A misconfiguration of a network device might make its behavior appear suspicious. Detecting such cases is beneficial on its own, as they may be security vulnerabilities. Another benefit of anomaly detection techniques is that they can be used as tools for creating signature-based detection rules. One way is to try to extract patterns from the detected anomalies, e.g., important features and their corresponding value ranges, which can be used to create signatures.

Arguably, one of the most promising application areas of ANIDS is monitoring IoT networks. Many IoT devices have a relatively simple network profile [45], which can be learned efficiently using anomaly detection techniques. Therefore, we can expect the ANIDS to be more precise and produce less false alarms in IoT network environments.

3.3 Related Algorithms

3.3.1 Deep Neural Networks (DNN)

Deep Neural Networks (DNNs) are a class of machine learning models inspired by the structure and function of the human brain, particularly the neural networks that make up the human nervous system. These networks consist of layers of interconnected nodes, also called neurons or artificial neurons, organized into input, hidden, and output layers.

Here's a breakdown of key concepts within Deep Neural Networks:

Neurons/Nodes:

Input Nodes: These nodes receive the initial input data. Each node represents a feature or attribute of the input.

Hidden Nodes: These nodes are part of one or more hidden layers and perform computations on the input data using weights and activation functions.

Output Nodes: The final layer produces the network's output. The number of output nodes depends on the type of problem (e.g., binary classification, multi-class classification, regression).

Weights and Connections:

Each connection between nodes is associated with a weight. These weights are parameters that the neural network learns during training. They determine the strength of the influence one node has on another.

Activation Functions:

Activation functions introduce non-linearity to the network, allowing it to learn from complex patterns. Common activation functions include sigmoid, tanh, and rectified linear unit (ReLU).

Layers:

DNNs consist of an input layer, one or more hidden layers, and an output layer. The term "deep" in deep learning refers to the presence of multiple hidden layers.

Feedforward and Backpropagation:

During the training phase, input data is fed forward through the network to produce an output. The output is compared to the actual target, and the difference (loss) is calculated. Backpropagation is then used to update the weights in the network, minimizing the loss and improving the model's performance. This process is often performed using optimization algorithms like gradient descent.

Training:

The neural network learns from a labeled dataset during the training phase. The goal is to minimize the difference between the predicted output and the actual target by adjusting the weights through backpropagation.

3.3.2 Recurrent Neural Networks

The intelligence of humans, as well as most animals, depends on having a memory of the past. This can be short-term, as when combining sounds to make words, and long-term, for example where the word ``she" can refer back to ``Anne" mentioned hundreds of words earlier. This is

exactly what RNN provides in neural networks. It adds feedback that enables using the outputs of previous time step while processing the current time-step input. It aims to add memory cells that function similarly to human long-term and short-term memories.

RNNs add recurrent layers to the NN (Neural Network) model. Figure 8 presents a generic model for RNNs that consists of three sets of layers (input, recurrent, and output). Input layers take the sensor output and convert it into a vector that conveys the features of the input. These are followed by the recurrent layers, which provide feedback. In most recent recurrent layer models, memory cells exist as well. Subsequently, the model completes similarly to most NN models with Fully Connected (FC) layers and an output layer that can be a softmax layer. FC layers and the output layer are grouped into the set of output layers in Figure 8.

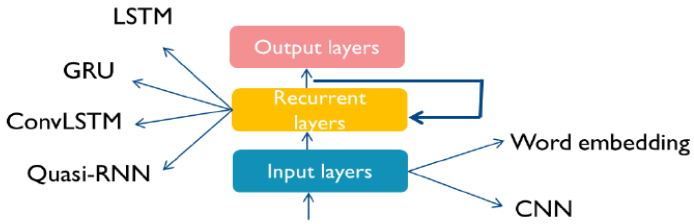

Figure 8: RNNs Presentation

A recurrent neural network looks very much like a feed-forward neural network, except it also has connections pointing backward. Let's look at the simplest possible RNN, composed of one neuron receiving inputs, producing an output, and sending that output back to itself, as shown in Figure 9 (left). At each time step t (also called a frame), this recurrent neuron receives the inputs x(t) as well as its own output from the previous time step, y(t–1). Since there is no previous output at the first time step, it is generally set to 0. We can represent this tiny network against the time axis, as shown in Figure 9 (right). This is called unrolling the network through time (it's the same recurrent neuron represented once per time step).

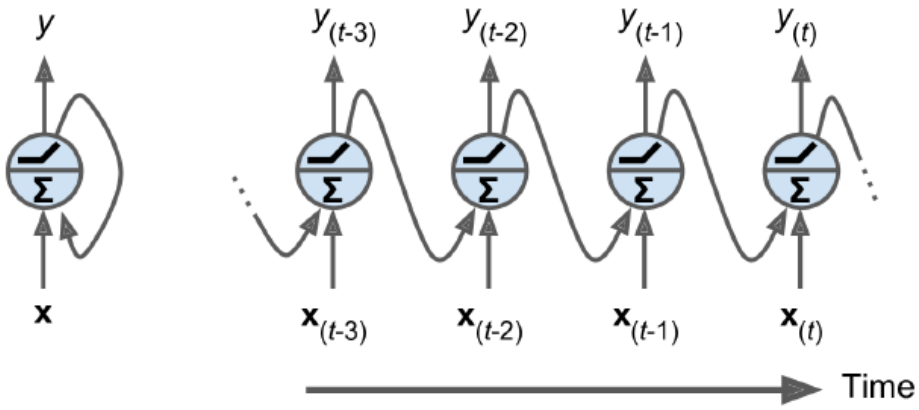

Figure 9: Structure of RNN

Each recurrent neuron has two sets of weights: one for the inputs x(t) and the other for the outputs of the previous time step, y(t–1). Let's call these weight vectors wx and wy. If we consider the whole recurrent layer instead of just one recurrent neuron, we can place all the weight vectors in two weight matrices, Wx and Wy. The output vector of the whole recurrent layer can then be computed pretty much as you might expect, as shown in Equation 1 (b is the bias vector and $\phi(\cdot)$ is the activation function (e.g.,ReLU).

Equation 1. Output of a recurrent layer for a single instance

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x^{\top}\mathbf{x}_{(t)} + \mathbf{W}_y^{\top}\mathbf{y}_{(t-1)} + \mathbf{b}\right)$$

Just as with feedforward neural networks, we can compute a recurrent layer's output in one shot for a whole mini-batch by placing all the inputs at time step t in an input matrix X(t) (see Equation 2).

Equation 2. Outputs of a layer of recurrent neurons for all instances in a minibatch

$$\begin{aligned}\mathbf{Y}_{(t)} &= \phi\left(\mathbf{X}_{(t)}\mathbf{W}_x + \mathbf{Y}_{(t-1)}\mathbf{W}_y + \mathbf{b}\right)\\ &= \phi\left(\left[\mathbf{X}_{(t)} \quad \mathbf{Y}_{(t-1)}\right]\mathbf{W} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix}\mathbf{W}_x\\ \mathbf{W}_y\end{bmatrix}\end{aligned}$$

Since the output of a recurrent neuron at time step t is a function of all the inputs from previous time steps, you could say it has a form of memory. A part of a neural network that preserves some state across time steps is called a memory cell (or simply a cell). A single recurrent neuron, or a layer of recurrent neurons, is a very basic cell, capable of learning only short patterns (typically about 10 steps long, but this varies depending on the task). Later in this chapter, we will look at some more complex and powerful types of cells capable of learning longer patterns (roughly 10 times longer, but again, this depends on the task).

In general a cell's state at time step t, denoted h(t) (the "h" stands for "hidden"), is a function of some inputs at that time step and its state at the previous time step: $h(t) = f(h(t–1), x(t))$. Its output at time step t, denoted y(t), is also a function of the previous

state and the current inputs. In the case of the basic cells we have discussed so far, the output is simply equal to the state, but in more complex cells this is not always the case.

Here are key components and concepts related to Recurrent Neural Networks:

Sequential Processing:

RNNs process input sequences one element at a time, maintaining a hidden state that captures information about previous inputs. This hidden state serves as a form of memory, enabling the network to learn patterns and relationships in sequential data.

Recurrent Connections:

RNNs have recurrent connections that allow information to persist over time. Each time the network receives a new input, it combines the current input with the information stored in the hidden state from the previous step.

Hidden State:

The hidden state in an RNN contains information about the network's understanding of the sequential data up to the current step. It serves as a memory that can capture long-term dependencies within the input sequence.

Training and Backpropagation Through Time (BPTT):

Backpropagation, in the context of recurrent neural networks (RNNs), is a key algorithm used for training the network to learn patterns and relationships in sequential data. RNNs are designed to handle sequences by maintaining internal states that capture information about previous elements in the sequence.

When we forward pass our input X_t through the network we compute the hidden state H_t and the output state O_t one step at a time. We can then define a loss function $L(O;Y)$ to describe the difference between all outputs O_t and target values Y_t as shown in Equation 5. This basically sums up every loss term of each update step so far. This loss term can have different definitions based on the specific problem (e.g. Mean Squared Error, Hinge Loss, Cross Entropy Loss, etc.).

a. Forward Pass

- During the forward pass, the RNN processes input sequences step by step, updating its internal state at each time step.

- At each time step t, the RNN takes the input x_t (which could be a word embedding, a feature vector, etc.) and the previous hidden state h_{t-1}, and computes the current hidden state ht using a set of weights and activation functions.

- The hidden state h_t is then used to compute the output y_t for the current time step using another set of weights and activation functions.

b. Backward Pass (Backpropagation Through Time, BPTT)[n]:

- After computing the output yt for each time step, the loss function is calculated comparing the predicted output to the actual target output.

- Starting from the last time step T, gradients of the loss with respect to the parameters (weights and biases) of the network are calculated using the chain rule of calculus.
- These gradients are then used to update the parameters of the network in order to minimize the loss function.
- The gradients are propagated backward through time, hence the term "Backpropagation Through Time (BPTT)[n].
- At each time step t, the gradients are also used to update the hidden state ht, taking into account the contributions of the error gradients from the future time steps (since the hidden state at time t depends on the hidden state at time t+1).

c. Gradient Clipping:

- To prevent exploding gradients, which can occur in RNNs due to the recurrent nature of the network, gradient clipping may be applied. This involves scaling the gradients if they exceed a certain threshold.

d. Repeat:

- Steps (a)-(c) are repeated for multiple epochs or until convergence, gradually improving the model's ability to capture patterns in the data.

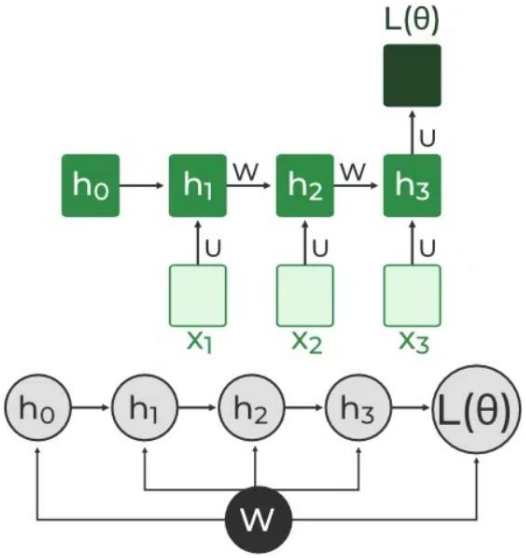

Figure 10: Backpropagation through time in RNN

L(θ)(loss function) depends on h3

h3 in turn depends on h2 and W

h2 in turn depends on h1 and W

h1 in turn depends on h0 and W

where h0 is a constant starting state.

Vanishing Gradient Problem:

As in most neural networks, vanishing or exploding gradients is a key problem of RNNs [12]. In Equation 1 and Equation 2 we can see Yt which basically introduces matrix multiplication over the (potentially very long) sequence, if there are small values (< 1) in the matrix multiplication this causes the gradient to decrease with each layer (or time step) and finally vanish [6]. This basically stops the contribution of states that happened far earlier than the current time step towards the current time step [6]. Similarly, this can happen in the opposite direction if we have large values (> 1) during matrix multiplication causing an exploding gradient which in result values each weight too much and changes it heavily [6]. This problem

motivated the introduction of the long short term memory units (LSTMs) to particularly handle the vanishing gradient problem. This approach was able to outperform traditional RNNs on a variety of tasks [6]. In the next section we want to go deeper on the proposed structure of LSTMs.

In this section, we cover the various types of recurrent layers. For each layer, we discuss the structure of the layer and the gate equations. The most popular recurrent layer is the Long Short Term Memory (LSTM) [39]. Changes have been proposed to the LSTM to enhance algorithmic efficiency or improve computational complexity. Enhancing algorithmic efficiency means improving the accuracy achieved by the RNN model.

Improving computational complexity means reducing the number of computations and the amount of memory required by an LSTM to run efficiently on a hardware platform. Techniques include LSTM with projection, and GRU which are discussed in upcoming sections, respectively. These changes can be applied to the gate equations, interconnections, or even the number of gates.

3.3.3 Long Short-Term Memory (LSTM)

Long Short-Term Memory is an improved version of recurrent neural network designed by Hochreiter & Schmidhuber. A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs address this problem by introducing a memory cell, which is a container that can hold information for an extended period. LSTM networks are capable of learning long-term dependencies in sequential data.The memory cell is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell. The input gate controls what information is added to the memory cell. The forget gate controls what information is removed from the memory cell. And

the output gate controls what information is output from the memory cell. This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.

Architecture and Working of LSTM

LSTM architecture has a chain structure that contains four neural networks and different memory blocks called cells. Information is retained by the cells and the memory manipulations are done by the gates. There are three gates –

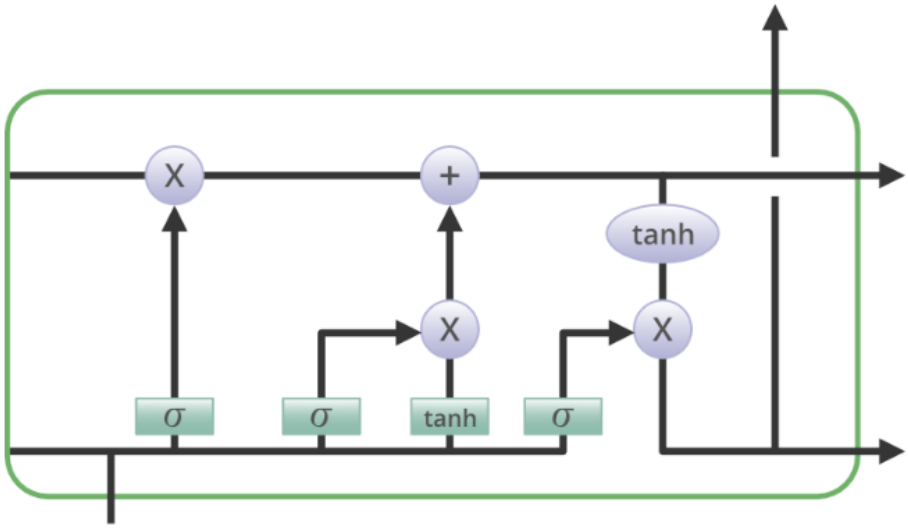

Figure 11: Architecture of LSTM

Forget Gate

The information that is no longer useful in the cell state is removed with the forget gate. Two inputs xt (input at the particular time) and ht-1 (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use. The equation for the forget gate is:

$f_t = \sigma(W_f \cdot [h_{\{t-1\}}, x_t] + b_f)$

where:

W_f represents the weight matrix associated with the forget gate.

$[h_{t-1}, x_t]$ denotes the concatenation of the current input and the previous hidden state.

b_f is the bias with the forget gate.

σ is the sigmoid activation function.

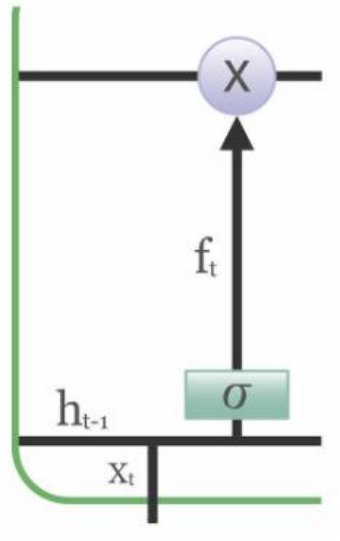

Figure 12: Forget gate of LSTM

Input gate

The addition of useful information to the cell state is done by the input gate. First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs h_t-1 and x_t. . Then, a vector is created using tanh function that gives an output from -1 to +1, which contains all the possible values from h_t-1 and x_t. At last, the values of the vector and the regulated values are multiplied to obtain the useful information. The equation for the input gate is:

$i_t = \sigma(W_i \cdot [h_{\{t-1\}}, x_t] + b_i)$

$\hat{C}_t = \tanh(W_c \cdot [h_{\{t-1\}}, x_t] + b_c)$

We multiply the previous state by ft, disregarding the information we had previously chosen to ignore. Next, we include it∗Ct. This represents the updated candidate values, adjusted for the amount that we chose to update each state value.

$C_t = f_t \odot C_{\{t-1\}} + i_t \odot \hat{C}_t$

where

$\odot$ denotes element-wise multiplication

tanh is tanh activation function

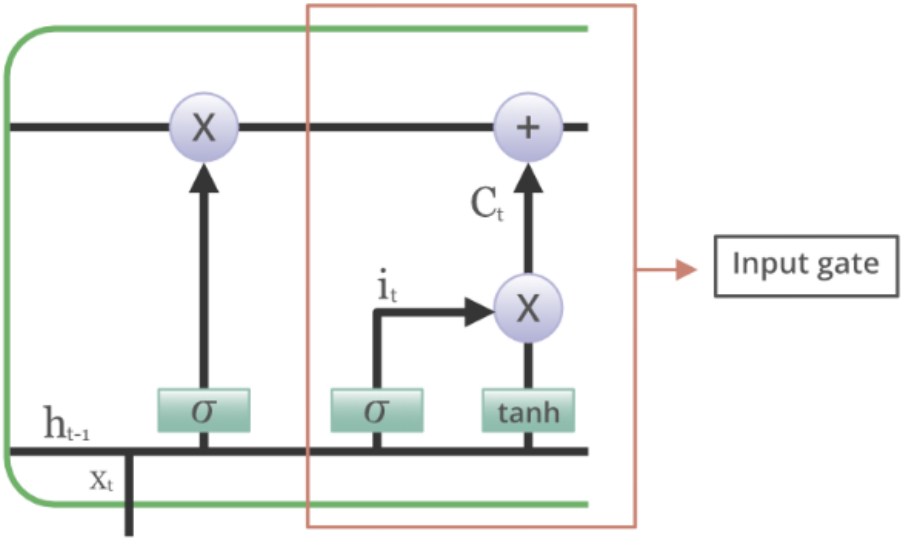

Figure 13: Input gate of LSTM

Output gate

The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs ht-1 and xt. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell. The equation for the output gate is:

$o_t = \sigma(W_o \cdot [h_{\{t-1\}}, x_t] + b_o)$

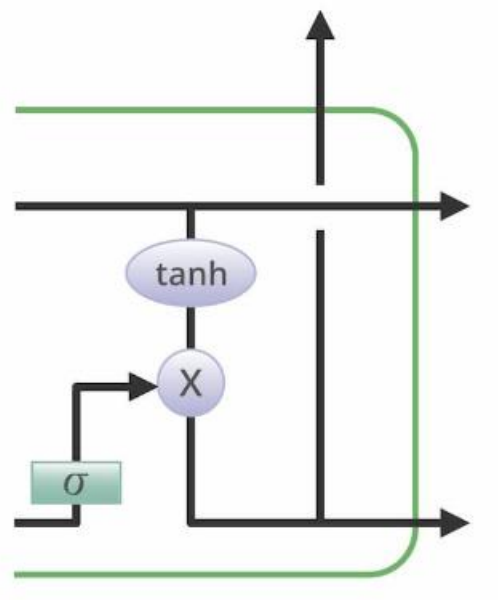

Figure 14: Output gate of LSTM

Advantages and Disadvantages of LSTM

The advantages of LSTM (Long-Short Term Memory) are as follows:

- Long-term dependencies can be captured by LSTM networks. They have a memory cell that is capable of long-term information storage.

- In traditional RNNs, there is a problem of vanishing and exploding gradients when models are trained over long sequences. By using a gating mechanism that selectively recalls or forgets information, LSTM networks deal with this problem.

- LSTM enables the model to capture and remember the important context, even when there is a significant time gap between relevant events in the sequence.

So where understanding context is important, LSTMS are used. eg. machine translation.

The disadvantages of LSTM (Long-Short Term Memory) are as follows:

- Compared to simpler architectures like feed-forward neural networks LSTM networks are computationally more expensive. This can limit their scalability for large-scale datasets or

constrained environments.

- Training LSTM networks can be more time-consuming compared to simpler models due to their computational complexity. So training LSTMs often requires more data and longer training times to achieve high performance.
- Since it is processed word by word in a sequential manner, it is hard to parallelize the work of processing the sentences.

3.3.4 Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that was introduced by Cho et al. in 2014 as a simpler alternative to Long Short-Term Memory (LSTM) networks.

The basic idea behind GRU is to use gating mechanisms to selectively update the hidden state of the network at each time step. The gating mechanisms are used to control the flow of information in and out of the network. The GRU has two gating mechanisms, called the reset gate and the update gate.

The reset gate determines how much of the previous hidden state should be forgotten, while the update gate determines how much of the new input should be used to update the hidden state.

The output of the GRU is calculated based on the updated hidden state.

The equations used to calculate the reset gate, update gate, and hidden state of a GRU are as follows:

Reset gate: $r_t = \text{sigmoid}(W_r * [h_{\{t-1\}}, x_t])$

Update gate: $z_t = \text{sigmoid}(W_z * [h_{\{t-1\}}, x_t])$

Candidate hidden state: $h_{t'} = \tanh(W_h * [r_t * h_{\{t-1\}}, x_t])$

Hidden state: $h_t = (1 - z_t) * h_{\{t-1\}} + z_t * h_{t'}$

where W_r, W_z, and Wh are learnable weight matrices, x_t is the input at time step t, $h_{\{t-1\}}$ is the previous hidden state, and h_t is the current hidden state.

In summary, GRU networks are a type of RNN that use gating mechanisms to selectively update

the hidden state at each time step, allowing them to effectively model sequential data.

To solve the Vanishing-Exploding gradients problem often encountered during the operation of a basic Recurrent Neural Network, many variations were developed. One of the most famous variations is the Long Short Term Memory Network(LSTM). One of the lesser-known but equally effective variations is the Gated Recurrent Unit Network(GRU).

Unlike LSTM, it consists of only three gates and does not maintain an Internal Cell State. The information which is stored in the Internal Cell State in an LSTM recurrent unit is incorporated into the hidden state of the Gated Recurrent Unit. This collective information is passed onto the next Gated Recurrent Unit. The different gates of a GRU are as described below:-

Update Gate(z): It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the Output Gate in an LSTM recurrent unit.

Reset Gate(r): It determines how much of the past knowledge to forget. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.

Current Memory Gate(\overline{h}_{t}): It is often overlooked during a typical discussion on Gated Recurrent Unit Network. It is incorporated into the Reset Gate just like the Input Modulation Gate is a sub-part of the Input Gate and is used to introduce some non-linearity into the input and to also make the input Zero-mean. Another reason to make it a sub-part of the Reset gate is to reduce the effect that previous information has on the current information that is being passed into the future.

The basic work-flow of a Gated Recurrent Unit Network is similar to that of a basic Recurrent Neural Network when illustrated, the main difference between the two is in the internal working within each recurrent unit as Gated Recurrent Unit networks consist of gates which modulate the current input and the previous hidden state.

Working of a Gated Recurrent Unit:

Take input the current input and the previous hidden state as vectors.

Calculate the values of the three different gates by following the steps given below:-
For each gate, calculate the parameterized current input and previously hidden state vectors by performing element-wise multiplication (Hadamard Product) between the concerned vector and the respective weights for each gate.
Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

Update Gate : Sigmoid Function
Reset Gate : Sigmoid Function

The process of calculating the Current Memory Gate is a little different. First, the Hadamard product of the Reset Gate and the previously hidden state vector is calculated. Then this vector is parameterized and then added to the parameterized current input vector.

To calculate the current hidden state, first, a vector of ones and the same dimensions as that of the input is defined. This vector will be called ones and mathematically be denoted by 1. First, calculate the Hadamard Product of the update gate and the previously hidden state vector. Then generate a new vector by subtracting the update gate from ones and then calculate the Hadamard Product of the newly generated vector with the current memory gate. Finally, add the two vectors to get the currently hidden state vector.

The above-stated working is stated as below:-

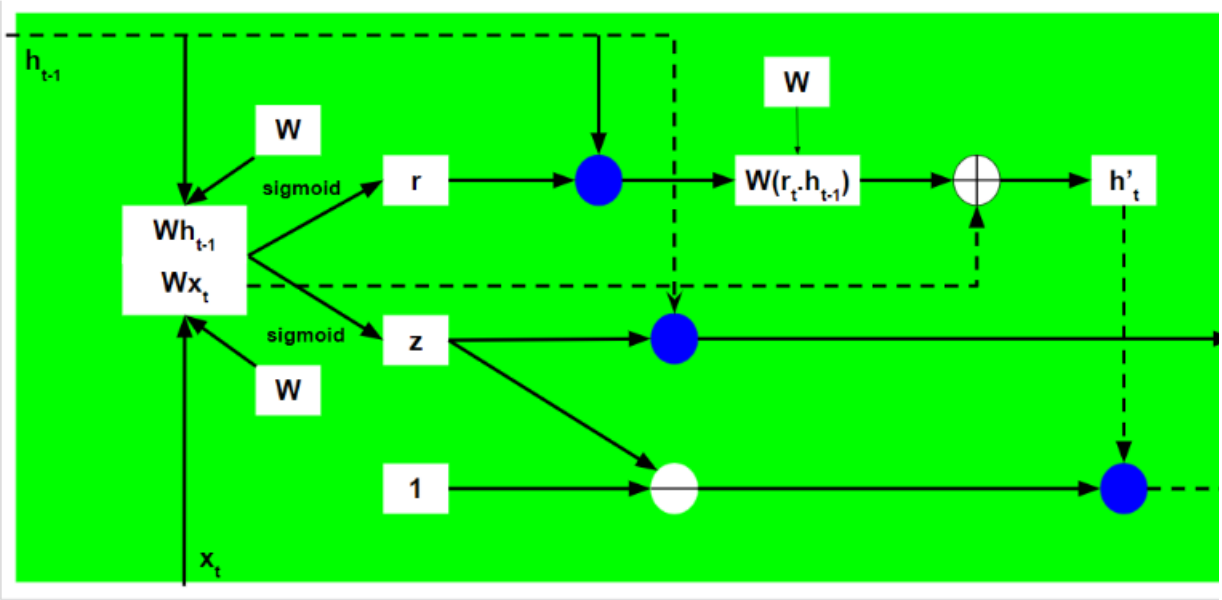

Figure 15: Working of GRU

Note that the blue circles denote element-wise multiplication. The positive sign in the circle denotes vector addition while the negative sign denotes vector subtraction(vector addition with negative value). The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.

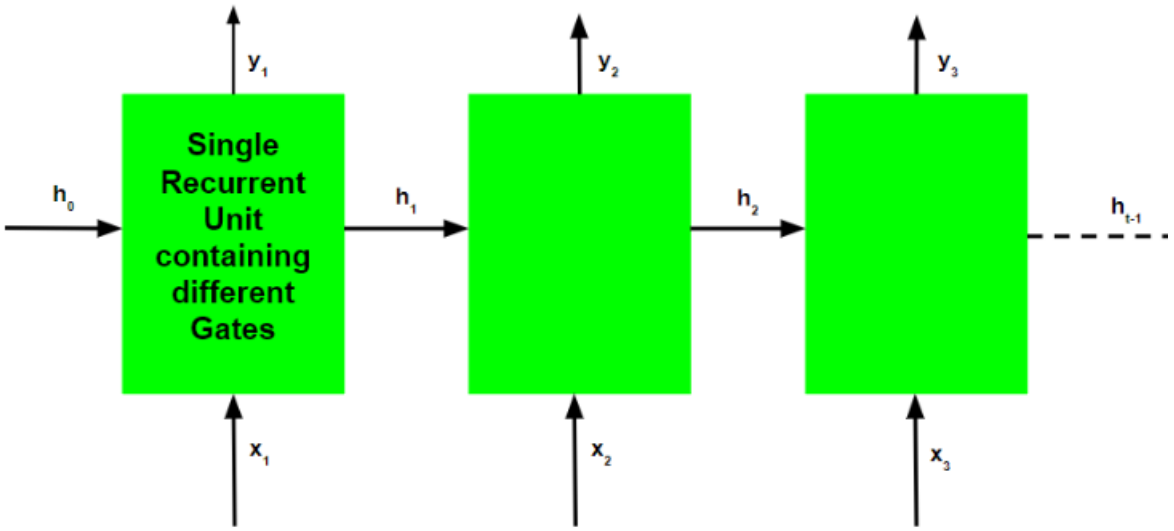

Figure 16: Comparison

Just like Recurrent Neural Networks, a GRU network also generates an output at each time step and this output is used to train the network using gradient descent.

3.3.5 Bi-directional Recurrent Neural Network (Bi-RNN)

An architecture of a neural network called a bidirectional recurrent neural network (BRNN) is made to process sequential data. In order for the network to use information from both the past and future context in its predictions, BRNNs process input sequences in both the forward and backward directions. This is the main distinction between BRNNs and conventional recurrent neural networks.

A BRNN has two distinct recurrent hidden layers, one of which processes the input sequence forward and the other of which processes it backward. After that, the results from these hidden layers are collected and input into a prediction-making final layer. Any recurrent neural network cell, such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit, can be used to create the recurrent hidden layers.

The BRNN functions similarly to conventional recurrent neural networks in the forward direction, updating the hidden state depending on the current input and the prior hidden state at each time step. The backward hidden layer, on the other hand, analyses the input sequence in the opposite manner, updating the hidden state based on the current input and the hidden state of the next time step.

Compared to conventional unidirectional recurrent neural networks, the accuracy of the BRNN is improved since it can process information in both directions and account for both past and future contexts. Because the two hidden layers can complement one another and give the final prediction layer more data, using two distinct hidden layers also offers a type of model regularisation.

In order to update the model parameters, the gradients are computed for both the forward and backward passes of the backpropagation through the time technique that is typically used to train BRNNs. The input sequence is processed by the BRNN in a single forward pass at inference time, and predictions are made based on the combined outputs of the two hidden layers.

layers.ventional recurrent neural networks in the forward direction, updating the hidden state depending on the current input and the prior hidden state at each time step. The backward hidden layer, on the other hand, analyses the input sequence in the opposite manner, updating the hidden state based on the current input and the hidden state of the next time step.

Compared to conventional unidirectional recurrent neural networks, the accuracy of the BRNN is improved since it can process information in both directions and account for both past and future contexts. Because the two hidden layers can complement one another and give the final prediction layer more data, using two distinct hidden layers also offers a type of model regularisation.

In order to update the model parameters, the gradients are computed for both the forward and backward passes of the backpropagation through the time technique that is typically used to train

BRNNs. The input sequence is processed by the BRNN in a single forward pass at inference time, and predictions are made based on the combined outputs of the two hidden layers. layers.

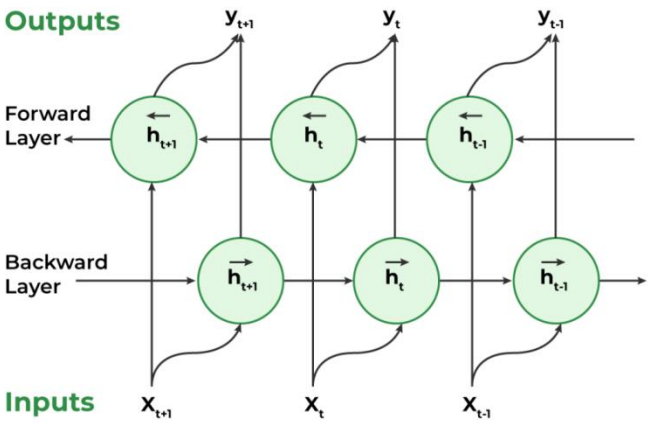

Figure 16: Architecture of Bi-Directional RNN

Working of Bidirectional Recurrent Neural Network

Inputting a sequence: A sequence of data points, each represented as a vector with the same dimensionality, are fed into a BRNN. The sequence might have different lengths.

Dual Processing: Both the forward and backward directions are used to process the data. On the basis of the input at that step and the hidden state at step t-1, the hidden state at time step t is determined in the forward direction. The input at step t and the hidden state at step t+1 are used to calculate the hidden state at step t in a reverse way.

Computing the hidden state: A non-linear activation function on the weighted sum of the input and previous hidden state is used to calculate the hidden state at each step. This creates a memory mechanism that enables the network to remember data from earlier steps in the process.

Determining the output: A non-linear activation function is used to determine the output at each step from the weighted sum of the hidden state and a number of output weights. This output has two options: it can be the final output or input for another layer in the network.

Training: The network is trained through a supervised learning approach where the goal is to minimize the discrepancy between the predicted output and the actual output. The network adjusts its weights in the input-to-hidden and hidden-to-output connections during training through backpropagation.

To calculate the output from an RNN unit, we use the following formula:

Ht (Forward) = A(X_t * WXH (forward) + H_{t-1} (Forward) * WHH (Forward) + bH (Forward)

Ht (Backward) = A(X_t * WXH (Backward) + H_{t+1} (Backward) * WHH (Backward) + bH (Backward)

where,

A = activation function,

W = weight matrix

b = bias

The hidden state at time t is given by a combination of Ht (Forward) and Ht (Backward). The output at any given hidden state is :

$Y_t = H_t$ * WAY + by

The training of a BRNN is similar to backpropagation through a time algorithm. BPTT algorithm works as follows:

Roll out the network and calculate errors at each iteration

Update weights and roll up the network.

However, because forward and backward passes in a BRNN occur simultaneously, updating the weights for the two processes may occur at the same time. This produces inaccurate outcomes.

Thus, the following approach is used to train a BRNN to accommodate forward and backward passes individually.

Advantages of Bidirectional RNN

- Context from both past and future: With the ability to process sequential input both forward and

backward, BRNNs provide a thorough grasp of the full context of a sequence.

- Enhanced accuracy: BRNNs frequently yield more precise answers since they take both historical and upcoming data into account.
- Efficient handling of variable-length sequences: When compared to conventional RNNs, which require padding to have a constant length, BRNNs are better equipped to handle variable-length sequences.
- Resilience to noise and irrelevant information: BRNNs may be resistant to noise and irrelevant data that are present in the data. This is so because both the forward and backward paths offer useful information that supports the predictions made by the network.
- Ability to handle sequential dependencies: BRNNs can capture long-term links between sequence pieces, making them extremely adept at handling complicated sequential dependencies.

Disadvantages of Bidirectional RNN

- Computational complexity: Given that they analyze data both forward and backward, BRNNs can be computationally expensive due to the increased amount of calculations needed.
- Long training time: BRNNs can also take a while to train because there are many parameters to optimize, especially when using huge datasets.
- Difficulty in parallelization: Due to the requirement for sequential processing in both the forward and backward directions, BRNNs can be challenging to parallelize.
- Overfitting: BRNNs are prone to overfitting since they include many parameters that might result in too complicated models, especially when trained on short datasets.
- Interpretability: Due to the processing of data in both forward and backward directions, BRNNs can be tricky to interpret since it can be difficult to comprehend what the model is doing and how it is producing predictions.

3.3.6 AutoEncoders

Autoencoders are a specialized class of algorithms that can learn efficient representations of input data with no need for labels. It is a class of artificial neural networks designed for unsupervised learning. Learning to compress and effectively represent input data without specific labels is the essential principle of an automatic decoder. This is accomplished using a two-fold structure that consists of an encoder and a decoder. The encoder transforms the input data into a reduced-dimensional representation, which is often referred to as “latent space” or “encoding”. From that representation, a decoder rebuilds the initial input. For the network to gain meaningful patterns in data, a process of encoding and decoding facilitates the definition of essential features.

Architecture of Autoencoder in Deep Learning

The general architecture of an autoencoder includes an encoder, decoder, and bottleneck layer.

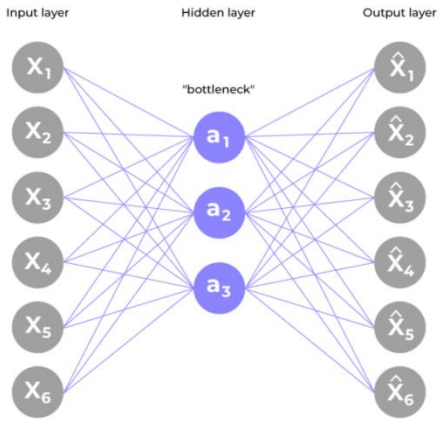

Figure 17: Architecture of Auto-Encoders

Encoder:

Input layer take raw input data

The hidden layers progressively reduce the dimensionality of the input, capturing important features and patterns. These layer compose the encoder.

The bottleneck layer (latent space) is the final hidden layer, where the dimensionality is significantly reduced. This layer represents the compressed encoding of the input data.

Decoder:

The bottleneck layer takes the encoded representation and expands it back to the dimensionality of the original input.

The hidden layers progressively increase the dimensionality and aim to reconstruct the original input.

The output layer produces the reconstructed output, which ideally should be as close as possible to the input data.

The loss function used during training is typically a reconstruction loss, measuring the difference between the input and the reconstructed output. Common choices include mean squared error (MSE) for continuous data or binary cross-entropy for binary data.

During training, the autoencoder learns to minimize the reconstruction loss, forcing the network to capture the most important features of the input data in the bottleneck layer.

After the training process, only the encoder part of the autoencoder is retained to encode a similar type of data used in the training process. The different ways to constrain the network are:

Keep small Hidden Layers: If the size of each hidden layer is kept as small as possible, then the network will be forced to pick up only the representative features of the data thus encoding the data.

Regularization: In this method, a loss term is added to the cost function which encourages the network to train in ways other than copying the input.

Denoising: Another way of constraining the network is to add noise to the input and teach the network how to remove the noise from the data.

Tuning the Activation Functions: This method involves changing the activation functions of various nodes so that a majority of the nodes are dormant thus, effectively reducing the size of

the hidden layers.

There are different types of autoencoders, but we used stacked or deep autoencoder in our thesis.

Stacked(Deep) Autoencoders

Just like other neural networks we have discussed, autoencoders can have multiple hidden layers. In this case they are called stacked autoencoders (or deep autoencoders). Adding more layers helps the autoencoder learn more complex codings. That said, one must be careful not to make the autoencoder too powerful. Imagine an encoder so powerful that it just learns to map each input to a single arbitrary number (and the decoder learns the reverse mapping). Obviously such an autoencoder will reconstruct the training data perfectly, but it will not have learned any useful data representation in the process (and it is unlikely to generalize well to new instances). The architecture of a stacked autoencoder is typically symmetrical with regard to the central hidden layer (the coding layer).

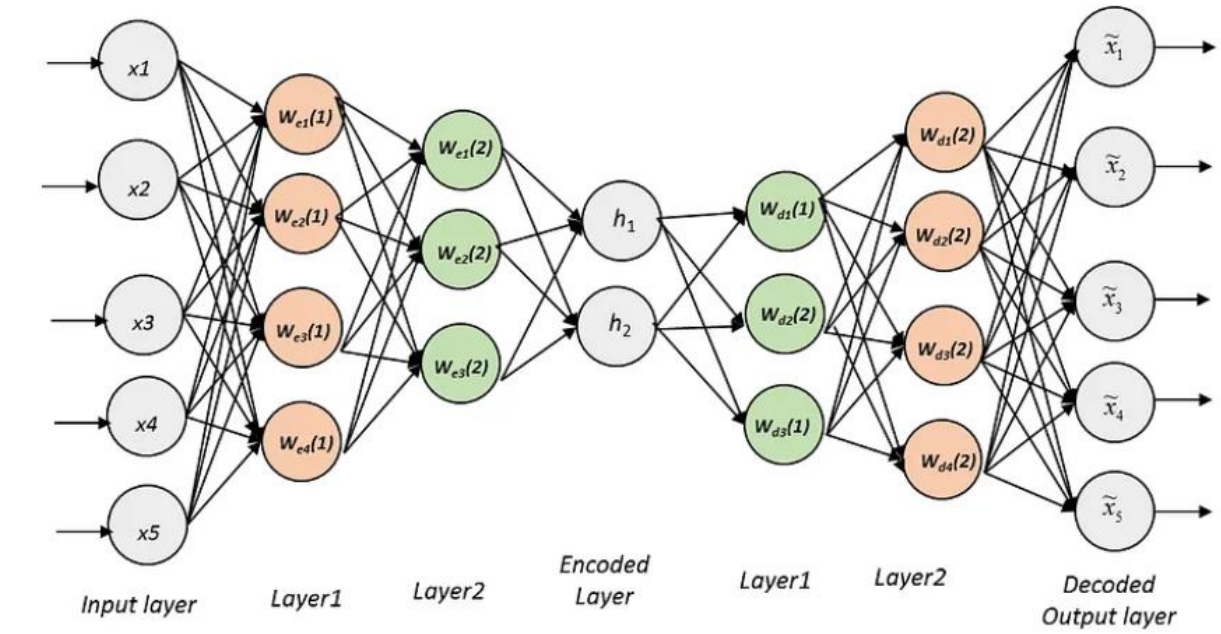

Figure 18: Architecture of Stacked Auto-Encoders

3.3.7 Transformer

A Transformer is a type of deep learning model architecture introduced in the paper "Attention is All You Need" by Vaswani et al. It has become widely popular, particularly in natural language processing (NLP) tasks, due to its ability to capture long-range dependencies in sequential data efficiently.

Here's a broad description of the Transformer architecture:

Self-Attention Mechanism: The core component of a Transformer is the self-attention mechanism, which allows the model to weigh the importance of different words in a sequence when encoding or decoding. It computes attention scores for each word in the input sequence relative to all other words, capturing dependencies regardless of their distance apart.

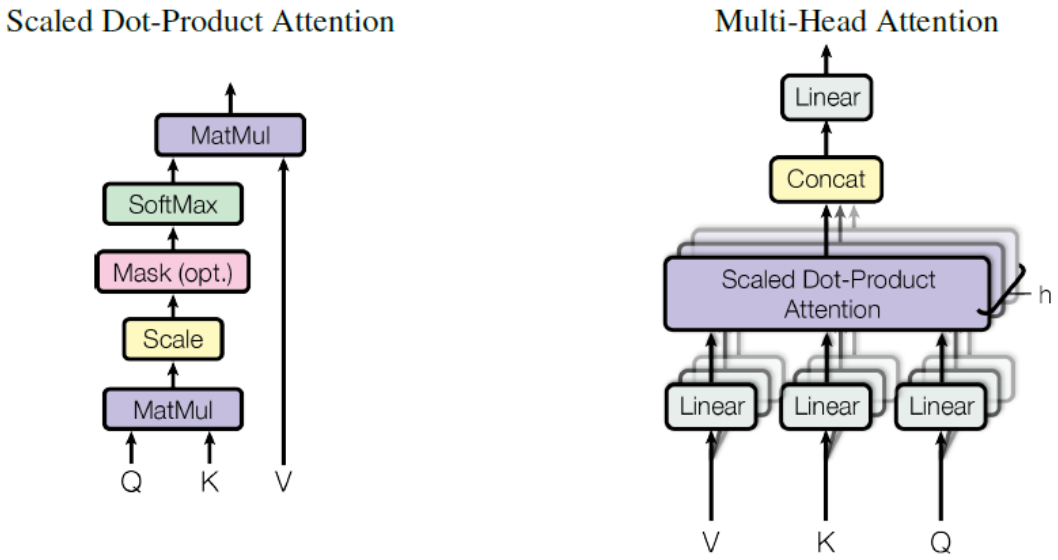

Figure 19: Attentions Mechanism

Encoder-Decoder Architecture: Transformers typically consist of an encoder and a decoder. The encoder processes the input sequence, while the decoder generates the output sequence. Each encoder and decoder layer contains multiple self-attention layers followed by feed-forward neural networks.

Positional Encoding: Since Transformers do not inherently understand the sequential order of

tokens, positional encoding is added to provide information about the position of each token in the sequence. This encoding is typically added to the input embeddings and allows the model to differentiate between tokens based on their position.

Feed-Forward Neural Networks: Both the encoder and decoder layers include feed-forward neural networks, which are applied independently to each position in the sequence. These networks consist of fully connected layers with non-linear activation functions, helping the model capture complex patterns in the data.

Residual Connections and Layer Normalization: Each sub-layer (self-attention and feed-forward networks) in the encoder and decoder contains residual connections followed by layer normalization. These techniques help with training deeper networks by mitigating the vanishing gradient problem and improving the flow of gradients during backpropagation.

Multi-Head Attention: To enhance the model's ability to focus on different parts of the input sequence simultaneously, multi-head attention mechanisms are used. This involves computing multiple sets of attention weights in parallel, each focusing on different parts of the input, and then concatenating the outputs.

Transformer Block: The basic building block of the Transformer architecture is the Transformer block, which consists of multiple layers of self-attention, feed-forward networks, and normalization. Stacking multiple Transformer blocks allows the model to capture complex patterns and dependencies in the data.

Overall, the Transformer architecture has demonstrated remarkable performance in various NLP tasks, such as machine translation, text summarization, and language modeling, and has also been adapted to other domains, including computer vision and speech processing.

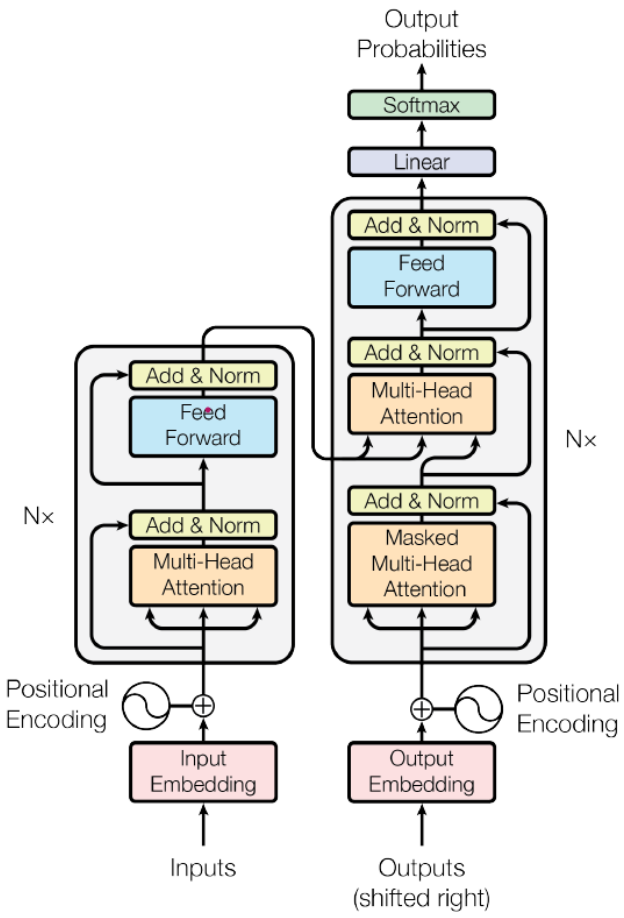

Figure 20: Architecture of Transformer

Comparison to Recurrent and Convolutional Layers

- Self-attention layers were found to be faster than recurrent layers for shorter sequence lengths and can be restricted to consider only a neighborhood in the input sequence for very long sequence lengths.

- The number of sequential operations required by a recurrent layer is based on the sequence length, whereas this number remains constant for a self-attention layer.

- In convolutional neural networks, the kernel width directly affects the long-term dependencies

that can be established between pairs of input and output positions. Tracking long-term dependencies would require using large kernels or stacks of convolutional layers that could increase the computational cost.

3.4 Network Traffic Data

There exist two standard formats for capturing network traffic data:

a) In the packet-based format, a data point is an IP packet, including its payload and header. Thus, it provides a complete picture of the information transferred in the network. It is a common belief that examining payload data may be necessary for detecting certain kinds of attacks. However, one issue of working with packet data is its sheer volume: it may be too resourcefully expensive to store and process data produced by a large network. Another problem is privacy, as payload data often contains confidential information.

b) In the flow-based format, In the flow-based format, a data point, called a network flow, only contains meta-information about a connection between network nodes.

Network flows have to be generated from the packet data using a special networking software tool called a flow collector. The flow collector aggregates IP packets belonging to a single connection and summarizes the information about the sequence of packets into a network flow record. For connectionless protocols, like UDP, a timeout value is used to determine the flow duration. The following five-tuple identifies a network flow: the source IP address, the destination IP address, the source port, the destination port, and the transport protocol [46].

It should be mentioned that there are two types of network flows: unidirectional and bidirectional. In the unidirectional case, the packets from a network node A to a network node B are aggregated into one flow record, and the response packets from B to A are aggregated into another flow record. In the bidirectional case, all the packets are aggregated into a single flow record. Table 4 shows common attributes of a network flow.

Depending on the configuration of the flow exporter, additional attributes may be added to flows, such as bytes per second, mean inter-arrival time of packets, TCP flags, and so on. As we can see, flow-based data typically can be represented as tabular data with mixed features.

Table 4 shows common attributes of a network flow. Depending on the configuration of the flow exporter, additional attributes may be added to flows, such as bytes per second, mean inter-arrival time of packets, TCP flags, and so on. As we can see, flow-based data typically can be represented as tabular data with mixed features.

#	Attributes/Features
1	Timestamp
2	Duration
3	Source IP
4	Destination IP
5	Source port
6	Destination port
7	Protocol
8	Number of bytes transferred
9	Number of packets transferred

Table 4: Common attributes of a network flow.

The advantages and disadvantages of working with packet- and flow-based network data is summarized in Table 5. In the recent years, flow-based techniques have become prevalent in research and industry [46]. Arguably the most important reason is that it is impracticable to collect and process packet data produced by a large network. Furthermore, it is easier to work with tabular data. For these reasons, we use network traffic data in the bidirectional flow format in our experiments.

Property	Format	
	Packet	Flow
Data completeness	+	-
Confidentiality	-	+
Storage costs	-	+
Processing costs	-	+

Table 5: Advantages and disadvantages of packet- and flow-based network data

3.4.1 Network Intrusion Dataset Properties

Selecting appropriate network intrusion datasets is crucial for the design and evaluation of anomaly detection techniques [5].

During this process, the following properties of network intrusion datasets should be taken into account [6, 3]:

a) Nature of data. These properties describe the format of the provided network traffic data. As discussed above, network traffic data can be available in the packet- or flow-based format. Depending on the flow collector used, flow-based data can contain different sets of features. Lastly, some attributes, like IP addresses and payloads, may be anonymized or even removed due to privacy reasons.

b) Network environment. These properties describe the network environment and how the data was collected. Data can be captured in a real network, or it can be emulated using special software. Furthermore, the properties of the captured data will depend on the network type. Examples of different network environments include university networks, internet service provider networks, industrial IoT networks, and so on.c) Attack types. These properties describe the present attacks and how they were executed.

Ideally, the dataset should contain a diverse and representative set of attacks. The way the attack traces were obtained is also important. For example, attacks can be simulated using a script, or they can be executed by researchers in a lab which arguably produces more realistic data.

d) Label availability. These properties describe the provided labels and how they were obtained. Accurately labeled data is crucial for evaluation. The ways of obtaining labels include conducting a manual inspection or using a signature based NIDS. The labeling process is often error-prone, and it is beneficial to have an estimate of the labeling accuracy. Another important aspect is the level of detail provided in the labels. In the basic case, the labels may only indicate malicious and benign data points. More detailed labels may provide specific information on performed malicious activities.

e) Data Volume. These properties describe the volume and duration of data. The volume is either specified as the total number of the data points (packets, flows) or the physical size in GB. In general, packet-based data has a much larger size compared to flow-based data. Duration refers to the period during which the network traffic data was collected. For studying periodical effects, the duration has to span a sufficient period (e.g., at least a month for comparing weekday vs. weekend patterns) [10].

Now the big question is, what are the properties we should look for to choose a perfect dataset? According to Ring et al. [6], it has to be “up-to-date, correctly labeled, and publicly available; and has to contain real network traffic with all kinds of attacks and normal user behavior and to span a long period of time.” Therefore, we have chosen the most recent dataset called "Iot-23" so that we get the most recent attacks type data with most varieties.

4 Experimental Set up:

4.1 Dataset:

We mainly focused on a labeled dataset called “Iot-23” . It is quite recent dataset of network traffic which was released in 2020[50]. The goal of the this dataset was to offer a large dataset of real and labeled IoT malware infections and IoT benign traffic for researchers to develop machine learning algorithms. It consists of 23 captures(also called scenarios), and in the 23 captures, there are 20 malicious captures and 3 benign captures.

Captures from infected devices will have the possible name of the malware sample executed on each scenario. The malware labels for IoT-23 dataset are: Attack, C&C, C&C-FileDownload. C&C-HeartBeat, C&C-HeartBeat-Attack, C&C-HeartBeat-FileDownload, C&C-Mirai, C&CTorii, DDoS, FileDownload, Okiru, Okiru-Attack, PartOfAHorizontalPortScan.

In addition, Zeek is a software that perform network analysing. The IoT-23 dataset we used is in the format of conn.log.labeled, which is the Zeek conn.log file that was generated from the Zeek network analyser using the original pcap file.

The variable types and definition for IoT-23 dataset are as shown in Table 1. Since the dataset is huge, we have decided to capture part of records from each individual dataset, then combine them to a new dataset. By doing this, our computer can handle the workload for the new dataset, and the new dataset remains most of the attack types of IoT-23 dataset.

ts	The time of the first packet
uid	A unique identifier of the connection
id	The connection's 4-tuple of endpoint addresses/ports
proto	The transport layer protocol of the connection
service	An identification of an application protocol
duration	How long the connection lasted
orig_bytes	The number of payload bytes the originator sent
resp_bytes	The number of payload bytes the responder sent
conn_state	The possible connection state values
local_orig	If the connection is originated locally, this will be T
local_resp	If the connection is responded locally, this will be T
missed_bytes	Indicate the number of bytes missed in content gaps
history	Records the state history of connections as a string
orig_pkts	Number of packets that the originator sent
orig_ip_bytes	Number of IP level bytes that the originator sent
resp_pkts	Number of packets that the responder sent
resp_ip_bytes	Number of IP level bytes that the responder sent
tunnel_parents	uid values for any encapsulating parent connections
orig_12_addr	Link-layer address of the originator

Table 6: Features and Definitions

Data Analysis

The rate of the attacks occurred through protocols type like TCP, ICMP and UDP .

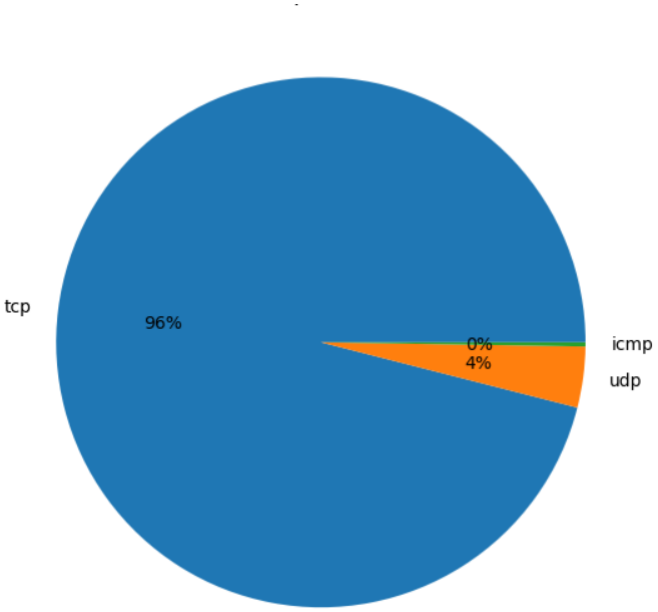

Figure 21: Attacks Percentage

Number of Intrusions access through each protocol. Here, benign means normal data.

proto	icmp	tcp	udp
label			
Attack	0	3915	0
Benign	3651	140561	53597
C&C	0	15100	0
C&C-FileDownload	0	43	0
C&C-HeartBeat	0	349	0
C&C-HeartBeat-FileDownload	0	8	0
C&C-Mirai	0	1	0
C&C-Torii	0	30	0
DDoS	0	138755	22
FileDownload	0	13	0
Okiru	0	262690	0
PartOfAHorizontalPortScan	0	825936	3

Table 7 : Attacks Number

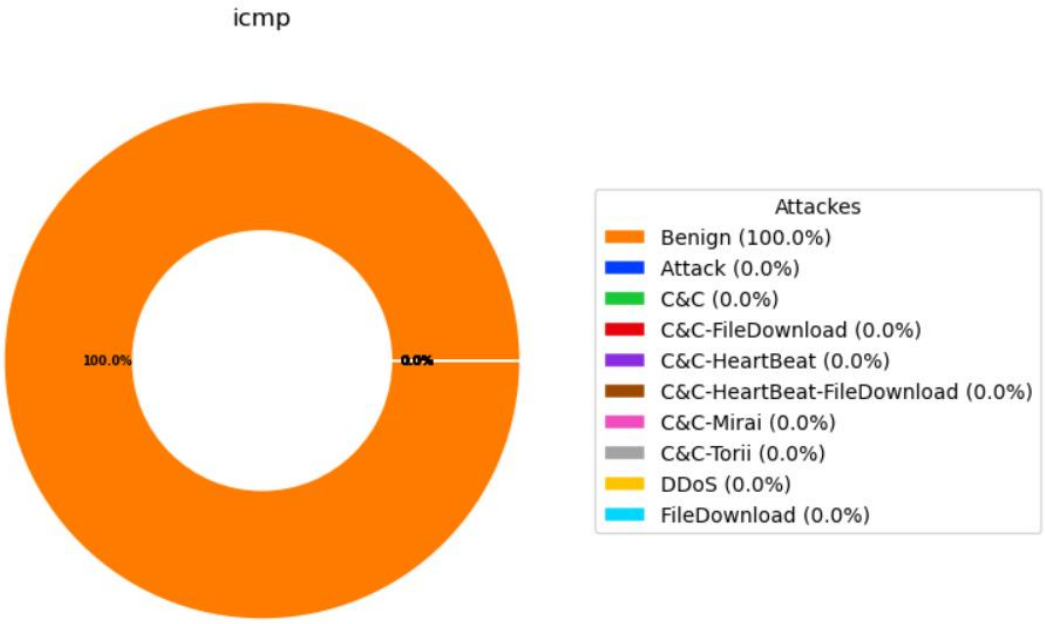

Figure 21: Label in icmp Protocol

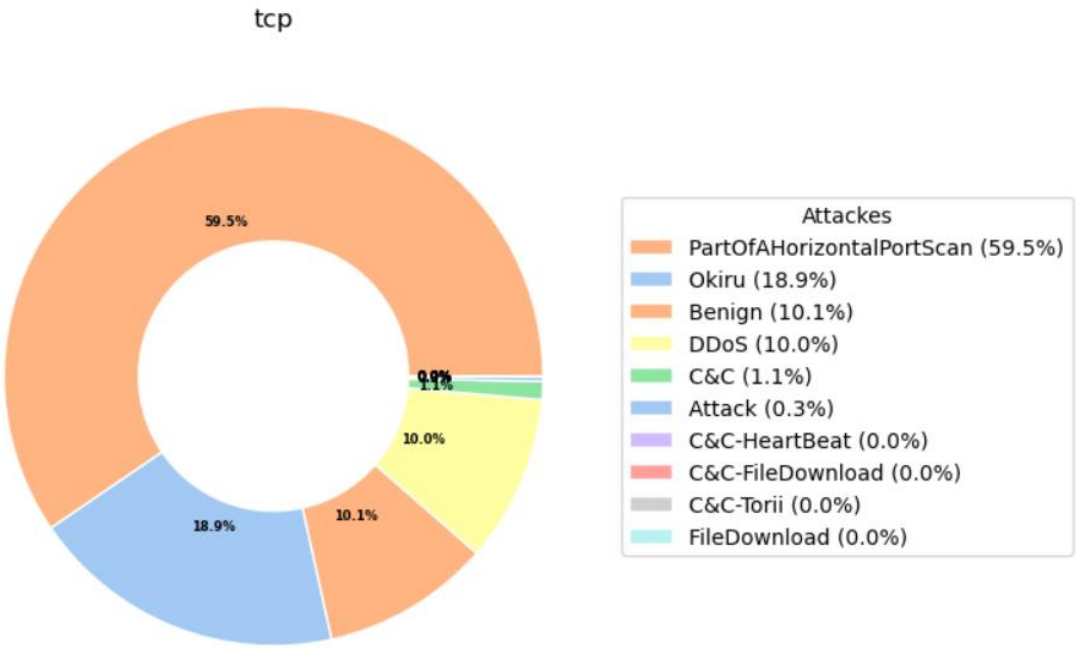

Figure 22: Label in tcp Protocol

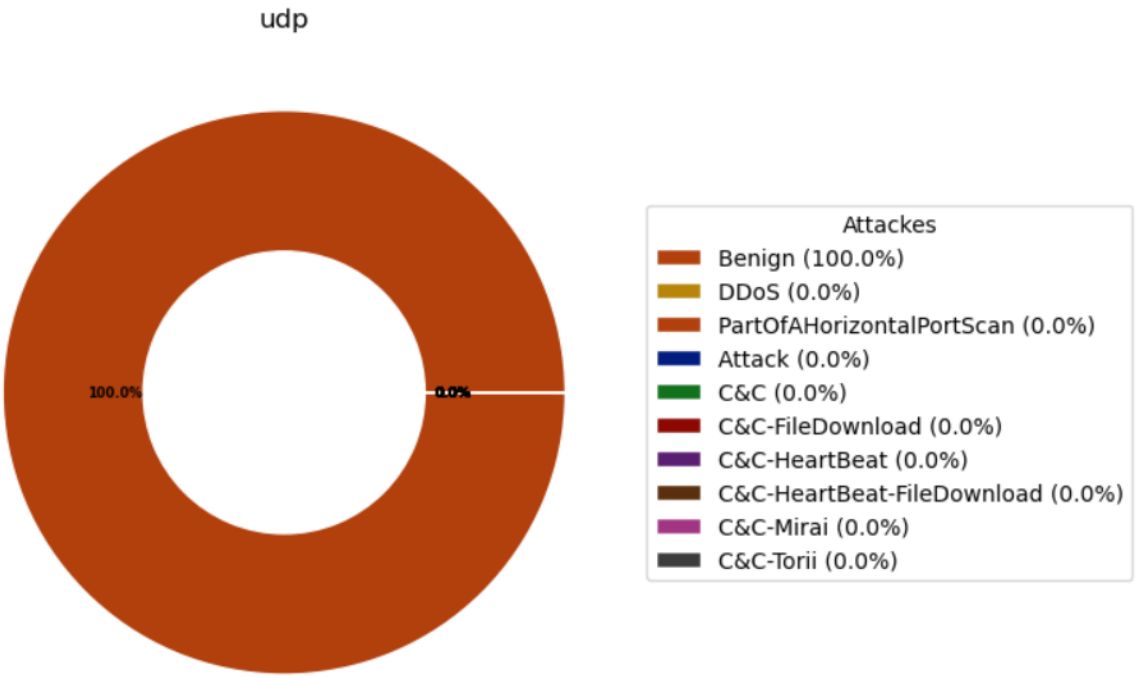

Figure 23: Label in udp Protocol

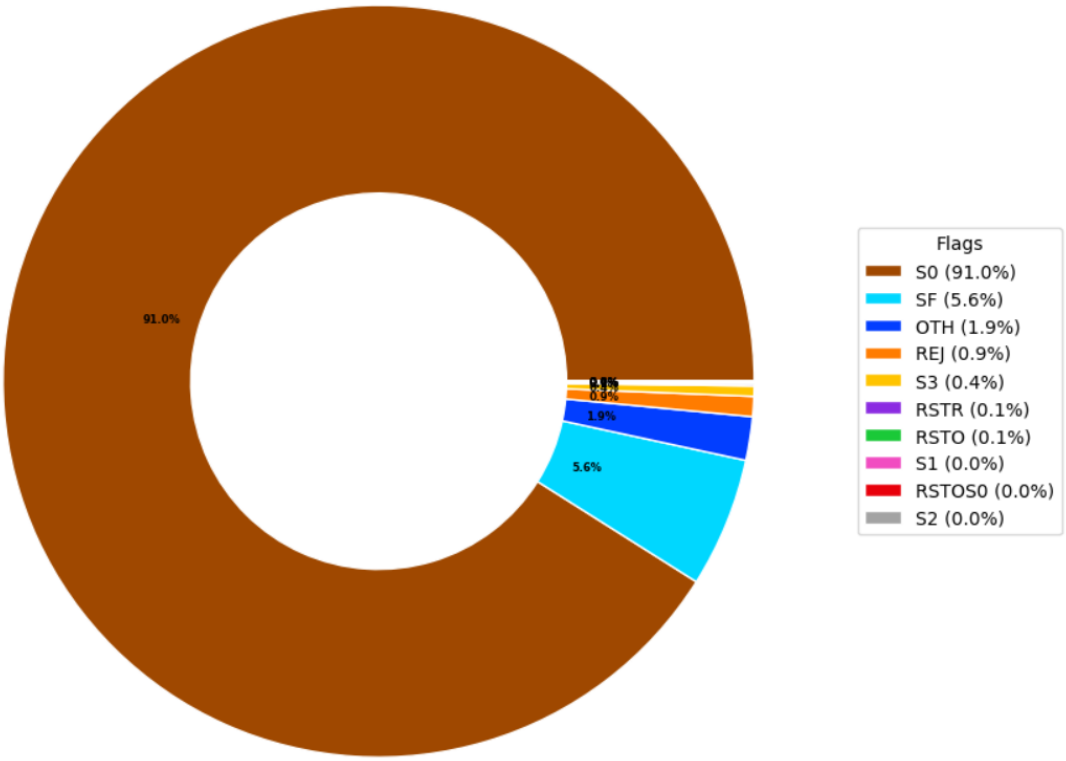

Figure 24: Connection State: Normal

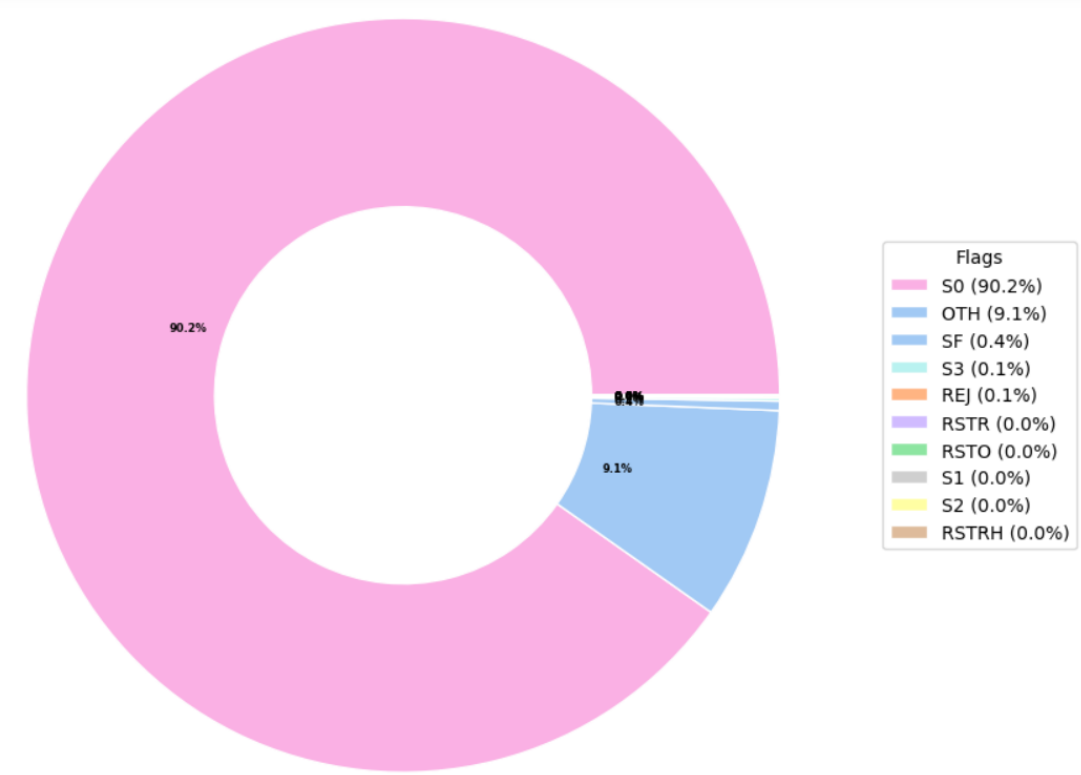

Figure 25: Connection State: Attack

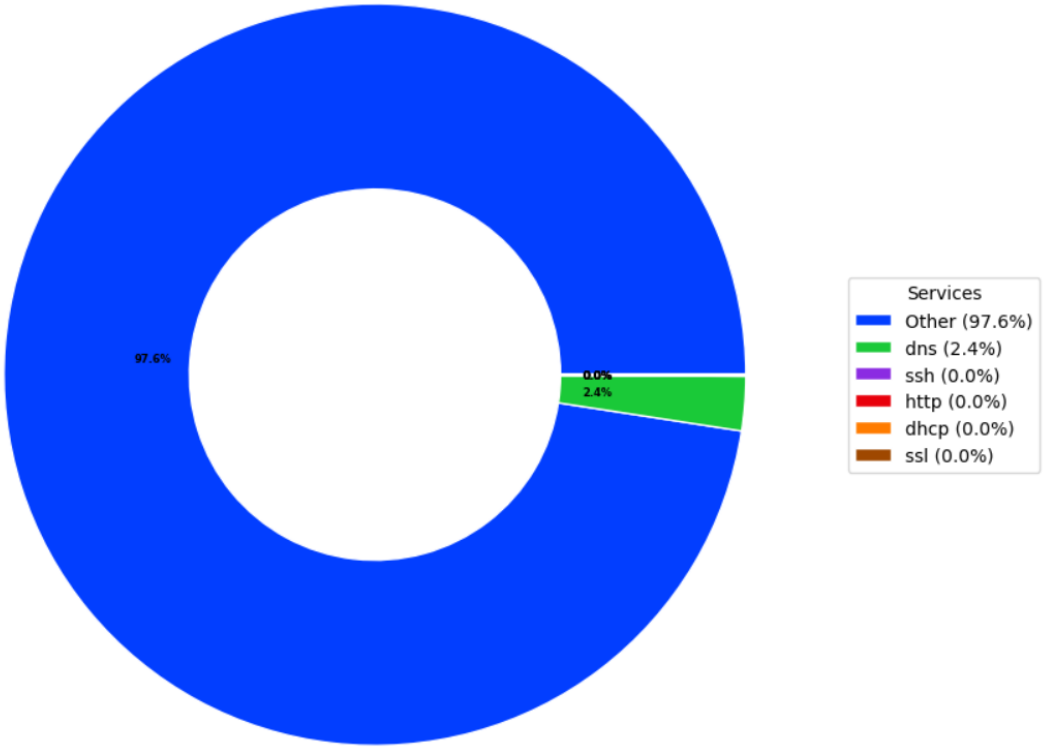

Figure 26: Service: Normal

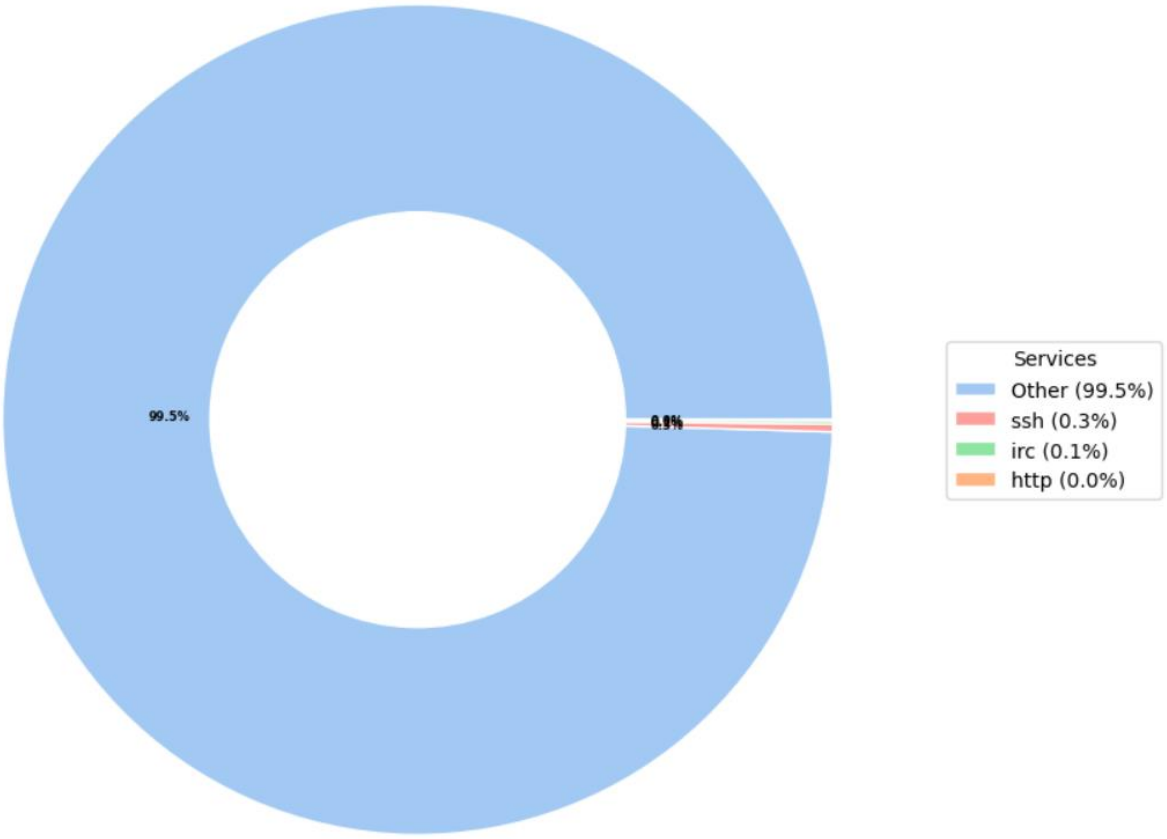

Figure 27: Service: Attack

Data Preprocessing

Features Selection:

First, we used the Python library Pandas to load all 23 datasets separately of the IoT-23 Dataset into data frames with a condition of skipping the first 10 rows and reading the one hundred thousand rows after. Then we combined all 23 data frames into a new data frame. Next, we dropped the variables that have no impact to the results. These variables are: ts, uid, id.orig h, id.orig p, id.resp h, id.resp p, service, local orig, local resp, history. Furthermore, we gave dummy values to the proto and conn state variables and replaced all the missing values with 0.

Last, the combined dataset is generated and saved as the iot23 combined.csv file. The iot23 combined.csv file contains a total of 1,444,674 records. Moreover, as shown in Table II, the combined file has 10 types of attack, including PartOfAHorizontalPortScan, Okiru, DDoS, Attack, C&C-HeartBeat, C&CFileDownload, C&C-Torii, FileDownload, C&C-HeartBeat-FileDownload, and C&C-Mirai.

Since we were interested in binary classification, we placed '1' (one) for all the attacks and zero ('0') for the benign rows in the "label" column using python 'map' function.

Then we analyzed each column and saw few columns have excessively one single dominated value than others are shown in Table 6. Therefore, we also dropped those columns as well.

Column	Unique Values	Value Counts
Conn_state_RSTO	0	1444521
	1	153
Conn_state_RSTOS0	0	1444644
	1	30
Conn_state_RSTR	0	1444123
	1	551
Conn_state_S2	0	1444647
	1	27
Conn_state_S3	0	1444217
	1	2457

Table 8: Stats of Dropped Columns

After dropping those columns, we got in total 15 column with label column.

The dataset is quite imbalanced based on the label column[Figure 28]

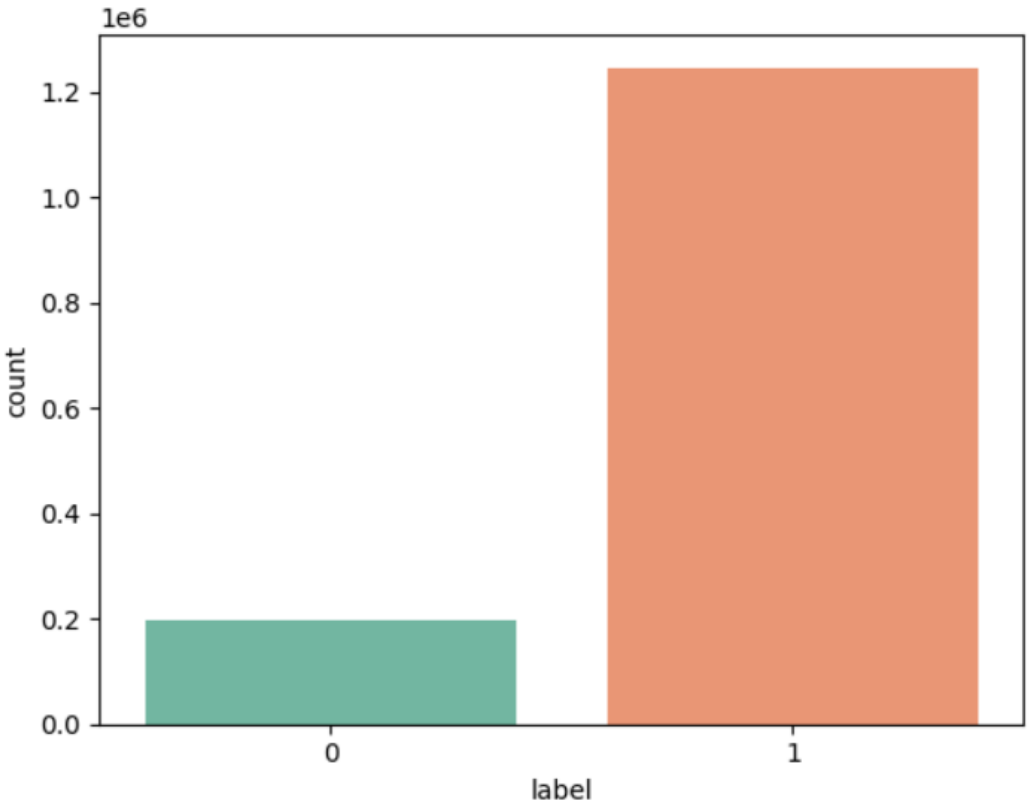

Figure 28: Ratio of Label

If you see the Figure 28 above, the attacking data are prevalent than benign data. If we apply an algorithm in this dataset, the prediction will be biased to the intrusion. Therefore, we have to increase the benign/normal data to balance the dataset because we don't want to lose any information.

In order to balancing the dataset without losing any information, we applied semi-supervised technique. First, we split the dataset into two portions: 1. Train (75%) and 2. Test (25%)

Then, we implemented a sequential deep-neural-network(DNN) of four layers with one output layer of one node (neuron) since it was a binary classification. We trained our network with the training portion of the dataset and the training accuracy was 89.3%.

Now our goal is to generate some normal random data.

1. In order to do that, we first make a dataset where only benign/normal rows are meaning we took all the rows where the value of the label column is ‘0’(zero) .

2. Then, we got the standard deviations of all the columns (14) using the ‘std()” function

3. After that, we multiply each standard deviation of each column to 0.1 to make the deviation value close and smaller, and convert to a ‘numpy’ array

4. we made a function named ‘random_val()” to generate random values applying normal distribution using ‘tf.random.normal’ function where

Seed value range: 1 to 256

Shape: 148534 rows and 14 columns

Mean: 0 (zero)

Standard Deviation: all the STDs we got from each column of normal data

5. And, finally we called our trained model to predict the benign data from the generated random values and through a loop we got the desired amount of benign data and concatenated with the original benign data. Thus, we got the quite balanced dataset show below[Figure 29].

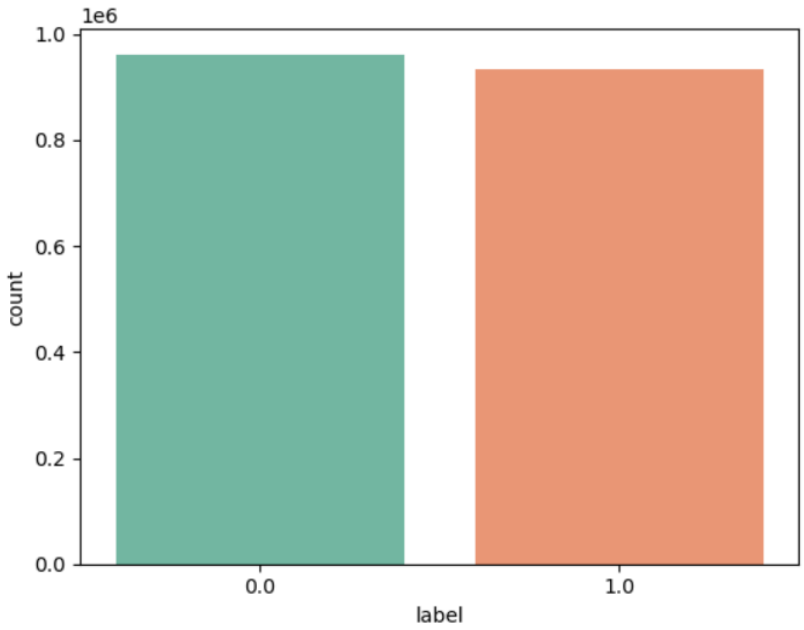

Figure 29: Ratio of Label

4.2 Methodolgies

Models Implementation:

Intrusions are a series of related malicious actions performed by an internal or external intruder that attempt to compromise the targeted system[1] . Since it is a series of malicious actions, we mainly focused on sequence learning. Therefore, we applied Recurrent Neural Network(RNN), Long-Short Term Memory(LSTM) and Gated Recurrent Units(GRU). But, we also tried Deep Neural Network(DNN). We always had tried to make an simple ,light-weighted networks due to overfitting and time. Because, more complex model usually has a prone to be overfitted and takes longer time to detect than a simpler model. Therefore, we implemented simpler models.

Deep Neural Networks(DNN): In our proposed sequential DNN, as shown in Table 2, we arrange one input layer, four dense layers and one output layer. The activation function for all the dense layers is Rectified Linear Unit (Relu) which is a linear function that fires the value if the weighted sum of the inputs in a neuron/node is positive otherwise it fires zero(0) and the

nodes will only be deactivated if the output of the weighted sum is less than 0 and in this way the 'relu' activation function works faster than others. And, since it is a binary classification task we set sigmoid function for the output layer, which ranges the output value from zero to one. The optimizer for the DNN model is adam, an optimized and popular gradient descent algorithm and loss function is binary_crossentropy. The total parameters for our model were 10,271 and all the parameters are trainable.

Layer (Type)	Output Shape	Number of parameters
Input(Dense)	(None, 100)	1500
Dense	(None, 50)	5050
Dense	(None, 40)	2040
Dense	(None, 40)	1640
Output(Dense)	(None, 1)	41
Total parameters: 10,271 Trainable parameters: 10,271 Non-trainable parameters: 0		

Table 9: Arch of DNN

Training Results	
Loss	Accuracy
0.1915	0.9086

Table 10: Training Accuracy of DNN

Recurrent Neural Networks (RNN): We implemented a simple sqquential RNN model of with few nodes in each layer like in first hidden layer we assign only 20 nodes/neurons because the accuracy gets down when we added more nodes like 40. We arrange two RNN layers and each layer, the default 'tanh' activation function is set.

For the Output Layer, we used a dense layer of one node with sigmoid activation function as we want a single output.

Layer (Type)	Output Shape	Number of parameters
Input(SimpleRNN)	(None, None, 20)	440
SimpleRNN	(None, 20)	820
Output(Dense)	(None, 1)	21
Total parameters: 1281 Trainable parameters: 1281 Non-trainable parameters: 0		

Table 11: Arch. of RNN

Training Results	
Loss	Accuracy
0.1230	0.9445

Table 12: Training Accuracy of RNN

But due to the unstable gradients and short term memory problem, we applied LSTM and GRU also.

Long-Short Term Memory (LSTM): We implement a very light-weighted sequential LSTM model where we used three LSTM layers with very few nodes. In the very first two layers, we assign only twenty and ten nodes and make sure the sequence return. However, in the last hidden layer, we assign only five nodes without returning the sequence since we will use dense layer in the last layer which is the output layer. Similarly, in the output layer we assign one node with sigmoid activation function using a dense layer.

Layer (Type)	Output Shape	Number of parameters
Input(LSTM)	(None, None, 20)	1760
LSTM	(None, None, 10)	1240
LSTM	(None, 5)	320
Output(Dense)	(None, 1)	6
Total parameters: 3326 Trainable parameters: 3326 Non-trainable parameters: 0		

Table 13: Arch. of LSTM

Training Results	
Loss	Accuracy
0.1503	0.9323

Table 14: Training Accuracy of LSTM

Gated Recurrent Unit (GRU): Here we arrange three layers. In the First hidden layer is a One Dimentional Convolutional where filter number is five, kernel size is two, stride is two and padding is valid. In the second hidden layer, we assign only five nodes and make sure the sequence return. In the third hidden layer, we assign only three nodes without returning the sequence. And, lastly in the output layer, one node with sigmoid activation using dense layer.

Layer (Type)	Output Shape	Number of parameters
Input(Conv1D)	(None, None, 5)	15
LSTM	(None, None, 5)	180
LSTM	(None, 3)	90
Output(Dense)	(None, 1)	4
Total parameters: 289 Trainable parameters: 289 Non-trainable parameters: 0		

Table 15: Arch. of GRU

As we can see, it has the least parameters compare to other like DNN, RNN and LSTM models. Thus, it is the most light-weighted model.

Training Results	
Loss	Accuracy
0.1047	0.9614

Table 16: Training Accuracy of GRU

Auto-encoding: For the autoencoding, we applied LSTM layers as the hidden layers since it is a sequence learning. In the encoding part, we assigned only ten nodes in the first layer where we make sure the sequence return and in the last layer of the encoding, we assigned only five nodes where we didn't make sure the sequence return. And, in the decoding part, as a latent representation, we applied RepeatVector layer where there are only 3 nodes and in the rest two layers there are ten and five nodes respectively.

Layer (Type)	Output Shape	Number of parameters
Input(LSTM)	(None, None, 10)	
LSTM	(None, None, 5)	
LSTM	(None, None, 3)	
RepeatVector	(None, 1)	
LSTM	(None, None, 5)	800
LSTM	(None, None, 10)	966
Output(Dense)	(None, 1)	
Total parameters: 1,766 Trainable parameters: 1,766 Non-trainable parameters: 0		

Table 17: Arch. of Autoencoders(LSTM)

Training Results	
Loss	Accuracy
0.1456	0.9355

Table 18: Training Accuracy of Autoencoders(LSTM)

We also implemented auto-encoding using GRU layer. That was similar as above but the loss was less.

Layer (Type)	Output Shape	Number of parameters
Input(GRU)	(None, None, 10)	
GRU	(None, None, 5)	
GRU	(None, None, 3)	
RepeatVector	(None, 1)	
GRU	(None, None, 5)	800
GRU	(None, None, 10)	966
Output(Dense)	(None, 1)	
Total parameters: 1,766 Trainable parameters: 1,766 Non-trainable parameters: 0		

Table 19: Arch. of Autoencoders(GRU)

Training Results	
Loss	Accuracy
0.1250	0.9436

Table 20: Training Accuracy of Autoencoders(GRU)

Bi-Directional RNN: In the first hidden layer, I used simple RNN layer with twenty nodes and for bidirectional process I used LSTM layer with ten nodes only.

Layer (Type)	Output Shape	Number of parameters
Input(SimpleRNN)	(None, None, 20)	440
Bidirectional(LSTM)	(None, None, 20)	2480
Output(Dense)	(None, 1)	21
Total parameters: 2,941 Trainable parameters: 2,941 Non-trainable parameters: 0		

Table 21: Arch. of Bi-RNN

A little bit improvement we notice than auto-encoding.

Training Results	
Loss	Accuracy
0.1223	0.9456

Table 22: Training Accuracy of Bi-RNN

Machine Learning & Deep Learning: We implemented an architecture where Deep Neural Networks (DNN) was applied for the feature extraction and Random Forest (RF) was used for the binary classification. We utilize DNN for the better understanding and representing of the complex or non-linear dataset. So that, the classifier (RF) will classify the intrusions more accurately. And, our hypothesize gets right. We get better accuracy than any other models in terms of False Positive and False Negative.

First of all, we split our dataset into two portion: one is for training which contains 75% of the dataset and another is for the testing which has 25% of the dataset. Then, we again split our training dataset in two groups: one is for the actual training (60%) and another is for the validation of the training (40%).

With the DNN, we train our model using the actual training data (60%) and predict using the validation of the training data (40%). So, the DNN predict the extracted features as a vector

shape which has ten dimensions. After that, we train our forest classifier with the extracted features and corresponding labeled column. Finally, we test our classifier with the test data (25%) and thus we get the better results.

DNN Architecture for the feature extraction: dimensions will be ten.

Layer (Type)	Output Shape	Number of parameters
Input(Dense)	(None, 100)	1500
Dense	(None, 50)	5050
Dense	(None, 40)	2040
Dense	(None, 10)	410
Total parameters: 9000 Trainable parameters: 9000 Non-trainable parameters: 0		

Table 23: Arch. of DNN

Transformer: A Transformer was initially applied to natural language processing tasks, abandoning the traditional recurrent neural network (RNN) and convolutional neural network (CNN) structures and solely relying on the attention mechanism to perform machine translation tasks, achieving excellent results. Compared to RNN-based sequential neural networks, the Transformer is superior. RNN training is iterative and sequential, resulting in particularly lengthy training times. In contrast, Transformer training is parallel, allowing all features to be trained simultaneously, dramatically increasing computational efficiency and reducing model training time. Therefore, in this paper, a Transformer was used as the base model to learn and extract features, thereby accelerating the model training speed.

The Transformer is composed of an encoder and a decoder, but for network intrusion detection tasks that do not require decoding, unlike sequence-to-sequence tasks such as machine

translation, only the encoder is needed to learn and extract features, which can be combined with "sigmoid" for classification. The Transformer classification structure is shown in Figure 30. The Transformer encoder consists of two sub-layers: multi-head attention mechanism and feed-forward neural network, with residual modules and normalization modules in each sub-layer. The multi-head attention mechanism allows the model to focus on different aspects of information, producing multiple subspaces that attend to different aspects of information, thus enhancing the model's performance.

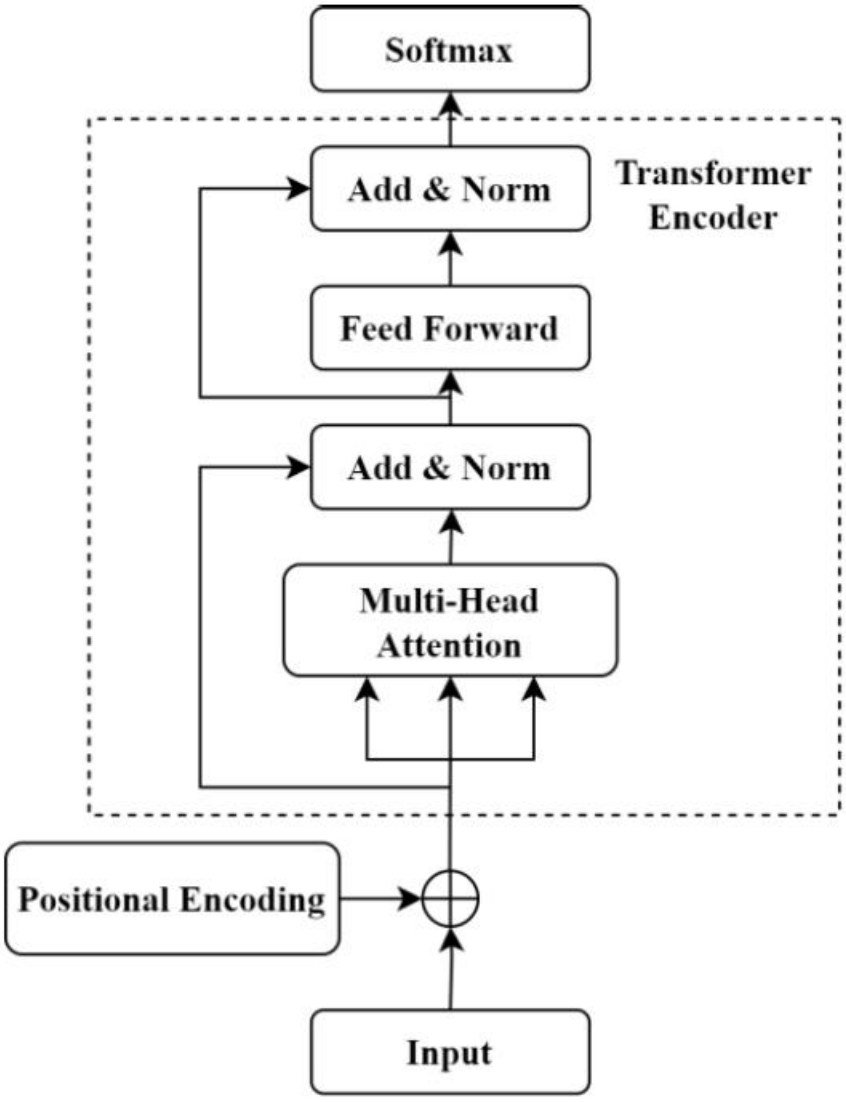

Figure 30: Design of Proposed Algo.

The Transformer model first encodes the input with a position encoding. The calculation formulas are shown as follows:

$$PE_{(pos,2i)} = \sin(pos/10,000^{2i/d_x})$$

$$PE_{(pos,2i+1)} = \cos(pos/10,000^{2i/d_x})$$

In the equation, the variable pos represents the position of a word within a text sequence sample, which is a value ranging from 0 to the maximum sequence length. dx denotes the dimension of the text encoding, while i is the index of a word within the encoding vector, ranging from 0 to dx. The location embedding function has a period that varies from 2π to 10,000* 2π, and each location in the encoding dimension is assigned a different combination of values of the sine and cosine functions with varying periods. This method generates distinct texture location data, which allows the model to capture the relationships between positions and the temporal features of natural language. By incorporating such information into the model, it can better capture the semantic meaning and structure of the text data, ultimately enhancing its performance in various natural language processing tasks.

In the context of natural language processing, machine translation involves encoding text-based input features. However, in intrusion detection tasks, the input features are typically numeric and do not require encoding. Encoding numeric features may alter the inherent information in the data and affect the amount of information that can be extracted by the model. Consequently, the model's ability to learn effectively from the features can be diminished, thereby reducing the classification performance.

It is crucial to carefully consider whether positional encoding is necessary for a particular type of data before applying it in a machine learning model. When applying positional encoding to text samples, encoding the text is a necessary step. In the positional encoding formula, the variable

pos represents the index of each word in the text sequence, while i represents the index of each element in the encoded vector of the word. The positional encoding method embeds the relationship between the feature vectors of each word in the text sequence, which represents the relationship between each word.

However, for intrusion detection samples, text encoding is not required, as intrusion detection samples mostly consist of numerical features. If the position is encoded according to the text samples, the variable pos in the positional encoding formula would represent the index of each sample, while i would represent the feature index of the sample. In this case, the positional encoding method would embed the relationship between each sample. However, in intrusion detection, the samples are independent of each other, and the embedded information would be irrelevant and invalid.

Although the samples in intrusion detection datasets are independent from each other, there is a correlation among their features. By embedding position codes that represent the position information of the features, the model can learn the dependency between feature positions and improve the learning performance. In other words, the model can capture the relationships between features in different positions, which can enhance its ability to learn and generalize.

After analyzing the above, this paper proposes an improved position encoding method for the Transformer of setting pos to 1 while letting i represent the positional index of each feature within a segment of samples. In the absence of pos, only i remains, and the encoding formula generated by the combination of values of sine and cosine functions with different periods will represent the positional information associated with i. As i represents the positional index of the feature, the proposed positional encoding embedding will capture the positional information of each feature, allowing the model to effectively learn the dependencies between feature positions. Consequently, the improved positional encoding formulas are shown as follows:

$$PE_{(pos,2i)} = \sin(1/10,000^{2i/d_x})$$

$$PE_{(pos,2i+1)} = \cos(1/10,000^{2i/d_x})$$

In the equation, dx represents the feature dimension of the sample and i represents the index of a feature within a segment of the sample, with i ranging from 0 to dx. The proposed position encoding method in this paper is more suitable for intrusion detection tasks than the previous approach. While the samples in intrusion detection datasets are independent, their features are often correlated. By enhancing the position encoding to include feature position information, the Transformer model can capture the positional dependencies between sample features, which increases the information available to the model and enhances its classification performance. The proposed method is therefore more appropriate for intrusion detection tasks, where feature correlations play an important role in identifying anomalies.

Evaluation Methodologies:

1) True Positives: The outcome where the model correctly predicts the positive class.

2) False Positives: The outcome where the model incorrectly predicts the positive class.

3) True Negative: The outcome where the model correctly predicts the Negative class.

4) False Negative: The outcome where the model incorrectly predicts the Negative class.

5) Precision: Precision is described as a measure of calculating the correctly identified positives in a model and is given by:

Precision =True Positives/(True Positives + False Positives)

5) Recall: It is a measure of actual number of positives that are correctly identified and is given by:

Recall =True Positives/(True Positives + False Negatives)

5 Experiments & Results

Testing Results:

NSL-KDD dataset: After applying PCA we get 8 features from 34

Classifiers	Accuracy
SVM(Linear)	91.2
SVM[RBF(C=10, gamma=0.5)]	96.9

Table 24: Performance on NSL-KDD dataset

CICIDS2017 dataset:

Recursive Feature Elimination(RFE) with cross validation(CV): We got most relevant 30 features and then apply random forest classification.

Models	Features	Accuracy
RFE	30	99.99

Table 25: Performance on CICIDS2017 dataset

Since the Iot-23 is the latest dataset, we mainly worked on this dataset.

To evaluate our models, we collected the dataset from the original source. Though there are several attacks with the names, We make the common choice of modeling benign network traffic as normal and attack traffic as anomalous. So that, our problem is binary classification.

benign = normal, attack = anomalous

Deep Neural Networks: Threshold>0.5

Overall Accuracy: 94.41%

	precision	recall	f1-score	support
0.0	1.00	0.89	0.94	240571
1.0	0.90	1.00	0.95	233610
accuracy			0.94	474181
macro avg	0.95	0.94	0.94	474181
weighted avg	0.95	0.94	0.94	474181

Figure 31: Stats of DNN

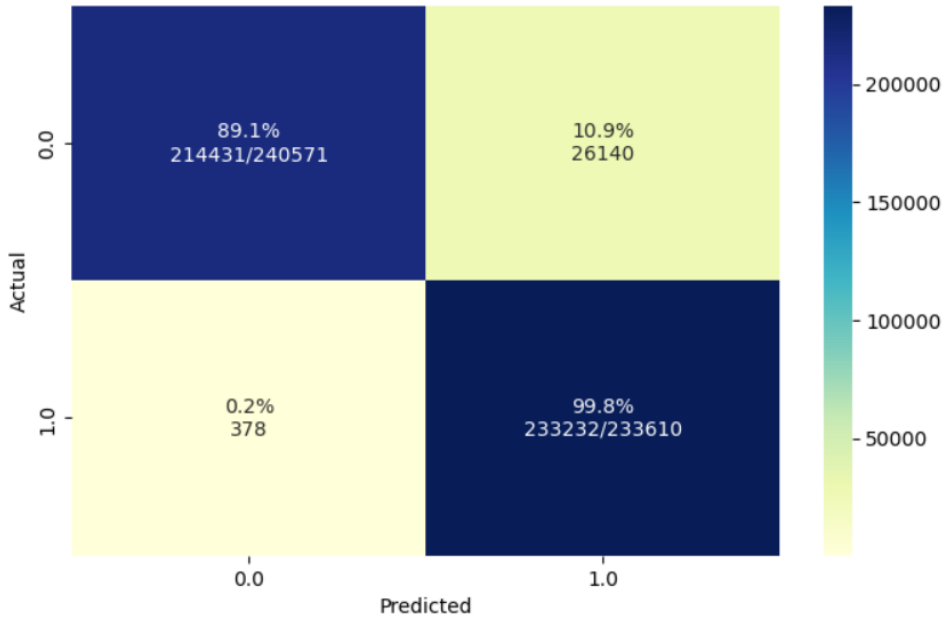

Figure 32: Stats of DNN

LSTM: Threshold >0.5

Overall Accuracy: 96.10%

	precision	recall	f1-score	support
0.0	0.99	0.93	0.96	240571
1.0	0.93	0.99	0.96	233610
accuracy			0.96	474181
macro avg	0.96	0.96	0.96	474181
weighted avg	0.96	0.96	0.96	474181

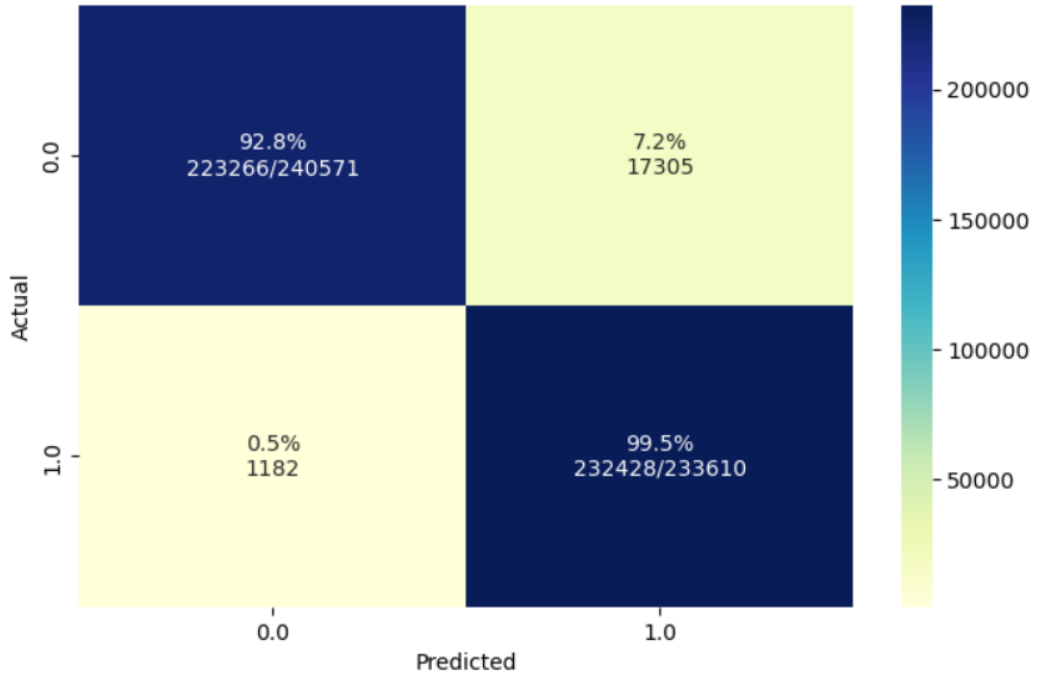

Figure 33: Stats of LSTM

GRU: Threshold >0.5

Overall Accuracy: 96.23%

	precision	recall	f1-score	support
0.0	1.00	0.93	0.96	240571
1.0	0.93	1.00	0.96	233610
accuracy			0.96	474181
macro avg	0.96	0.96	0.96	474181
weighted avg	0.96	0.96	0.96	474181

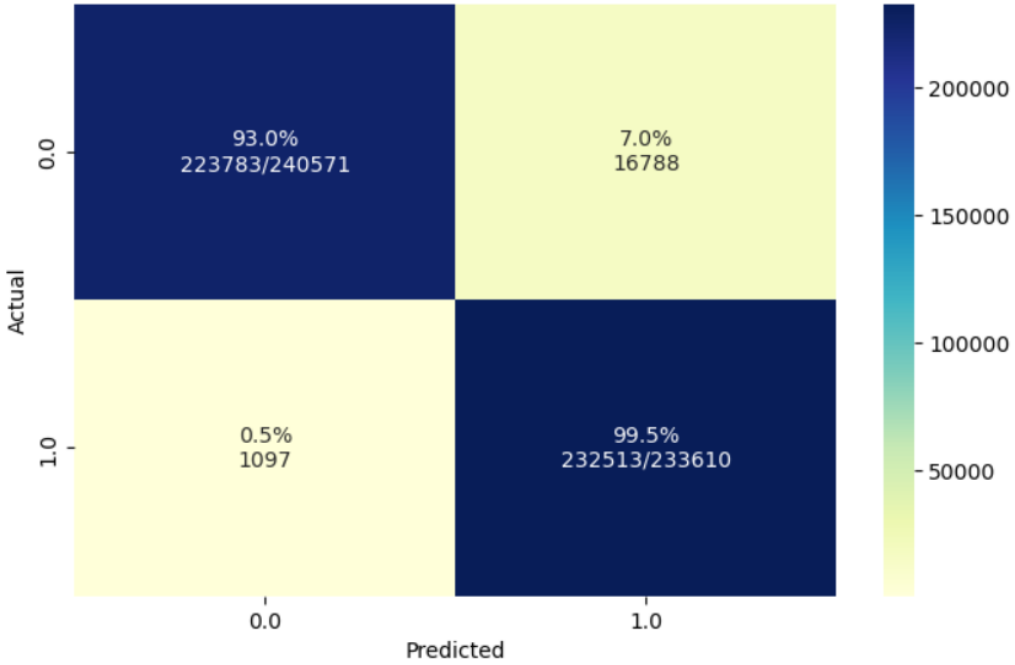

Figure 34: Stats of GRU

LSTM+AE: Threshold >0.5

Overall Accuracy: 94.39%

	precision	recall	f1-score	support
0.0	1.00	0.89	0.94	240571
1.0	0.90	1.00	0.95	233610
accuracy			0.94	474181
macro avg	0.95	0.94	0.94	474181
weighted avg	0.95	0.94	0.94	474181

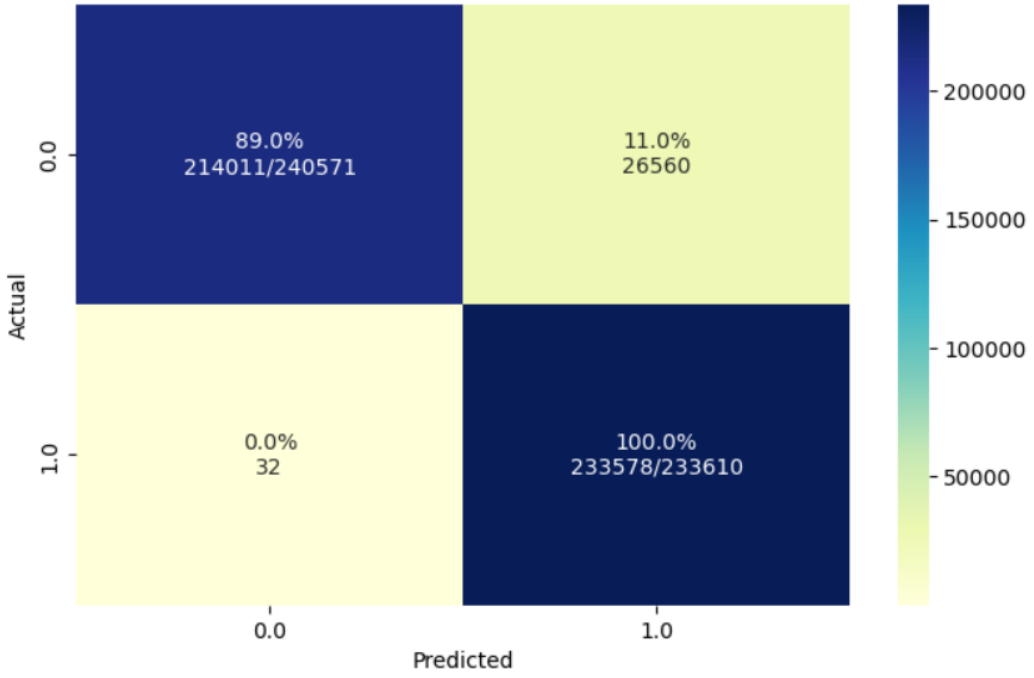

Figure 35: Stats of LSTM+AE

GRU+AE: Threshold >0.5

Overall Accuracy: 94.44%

	precision	recall	f1-score	support
0.0	1.00	0.89	0.94	240571
1.0	0.90	1.00	0.95	233610
accuracy			0.94	474181
macro avg	0.95	0.95	0.94	474181
weighted avg	0.95	0.94	0.94	474181

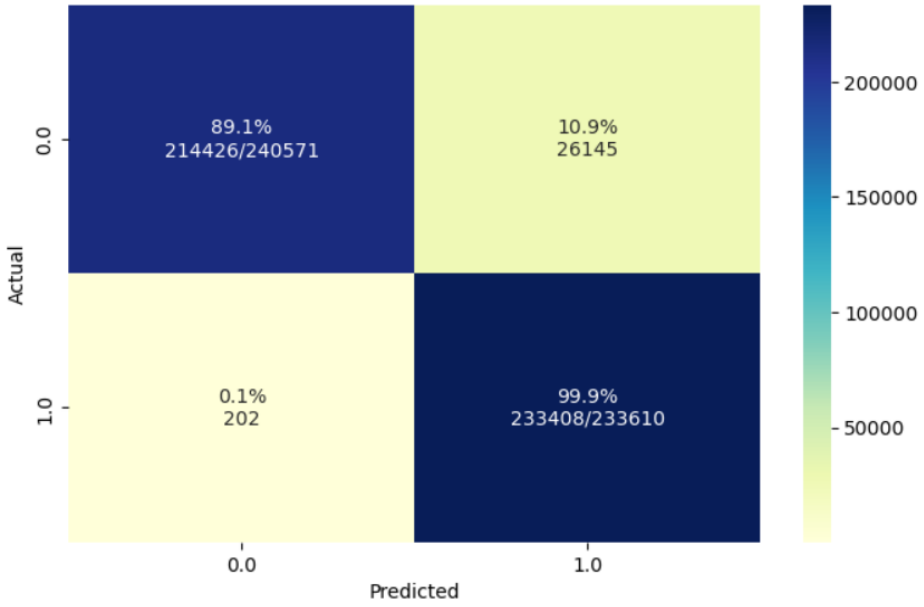

Figure 36: Stats of GRU+AE

Bi-Directional RNN: Threshold >0.5

Overall Accuracy: 94.47%

	precision	recall	f1-score	support
0.0	1.00	0.89	0.94	240571
1.0	0.90	1.00	0.95	233610
accuracy			0.94	474181
macro avg	0.95	0.95	0.94	474181
weighted avg	0.95	0.94	0.94	474181

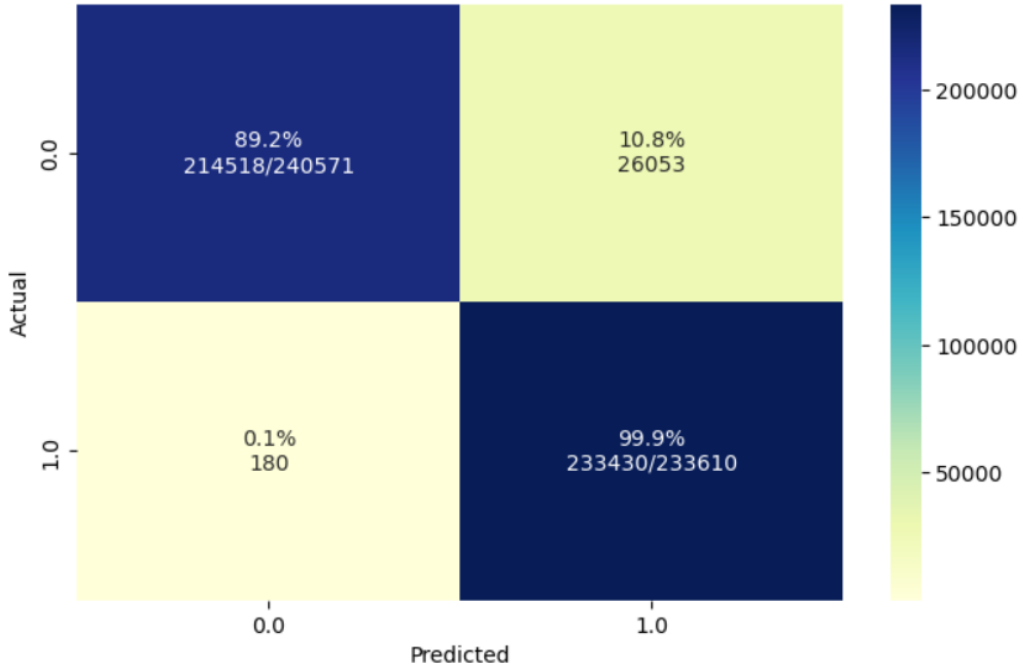

Figure 37: Stats of B-RNN

ML and DNN:

Overall Accuracy: 96.21%

	precision	recall	f1-score	support
0.0	0.99	0.93	0.96	240571
1.0	0.93	0.99	0.96	233610
accuracy			0.96	474181
macro avg	0.96	0.96	0.96	474181
weighted avg	0.96	0.96	0.96	474181

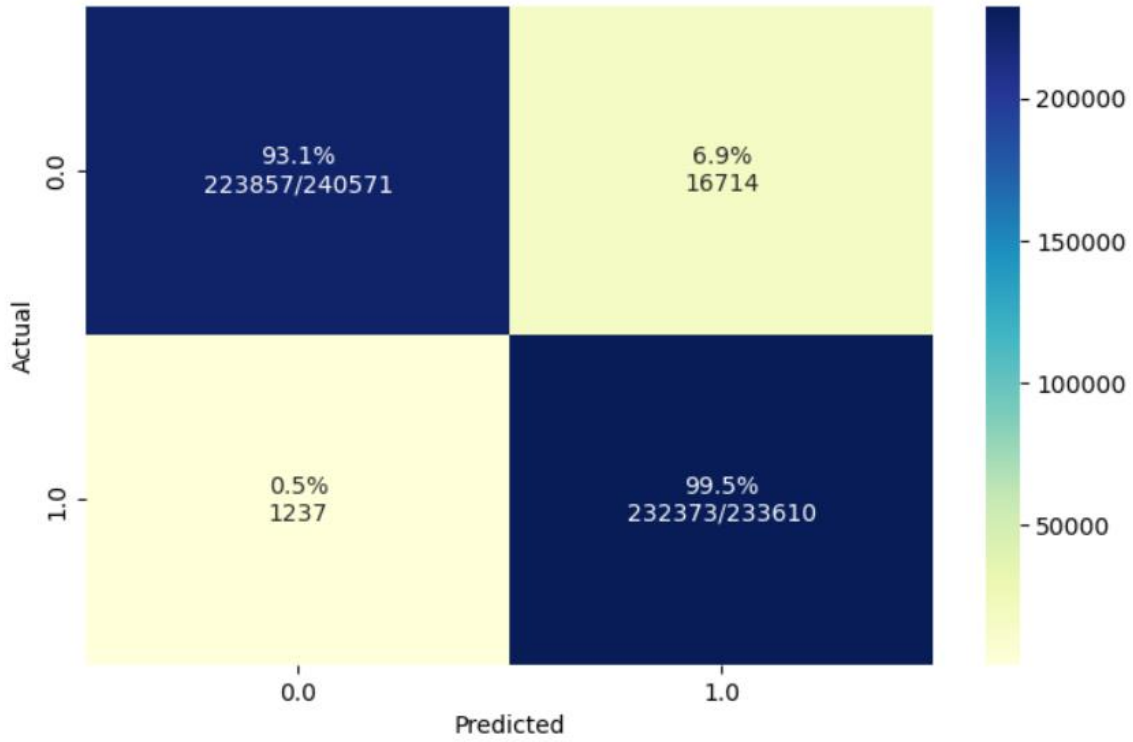

Figure 38: Stats of Ml & DNN

Transformer: In this time our this model don't work, we only got 50% accuracy but I hope after a little bit more optimization it will work better. We may try a more simpler architecture.

Comparisons

Comparing the performance among our models.

Model (Thr>0.5)	Accuracy	Label	Precision	Recall	False Positive (%)	False Negative (%)
DNN	94.41	0	1.00	0.89	10.9	0.2
		1	0.90	1.00		
LSTM	96.10	0	0.99	0.93	7.2	0.5
		1	0.93	0.99		
GRU	96.23	0	1.00	0.93	7.0	0.5
		1	0.93	1.00		
LSTM & AE	94.39	0	1.00	0.89	11.0	0.0
		1	0.90	1.00		
GRU & AE	94.44	0	1.00	0.89	10.9	0.1
		1	0.90	1.00		
Bi-Directional RNN	94.47	0	1.00	0.89	10.8	0.1
		1	0.90	1.00		
ML & DNN	96.21	0	0.99	0.93	6.9	0.5
		1	0.93	0.99		

Table 26: Performance Comparison on IoT-23 testing set

Comparing the performance among others.

Best Method	Highest Accuracy(%)
Support Vector Machine(SVM) [47]	69
Isolation Forest [48]	72
Random Forest[51]	89.7
Multi-layer Perceptron [52]	90.3
Few-Shot Learning[49]	93
Ours(GRU)	96.23

Table 27: Performance evaluation on IoT-23 testing set

6 Discussion:

To address the three issues of slow model training, imbalanced datasets, and poor binary-class detection performance in intrusion detection models, this paper proposes multiple models. Through multiple experiments, we have demonstrated the effectiveness of the proposed models in addressing these issues, not only reducing the model's features number but also enhancing its classification detection ability. Despite the notable improvements in detection accuracy attained by the proposed models, there remain certain limitations that warrant attention. Specifically, the accurate detection capability of the model, although enhanced, still requires further development. Future research endeavors may include the exploration of alternative strategies to enhance the binary classification performance of the proposed models. This may involve investigating diverse model architectures, such as incorporating attention mechanisms or exploring novel loss functions that are better equipped to capture the unique characteristics of the dataset. Additionally, efforts could be made to improve the interpretability of the model by analyzing the attention weights of the models and identifying significant features for intrusion detection. These approaches may culminate in further improvements in the overall detection performance and can have far-reaching implications beyond the scope of intrusion detection. Overall, this study contributes to the knowledge system by proposing several with novel approachs and demonstrating its effectiveness in addressing the three issues in intrusion detection models, while also emphasizing the need for further research and improvement in this area.

Conclusion

Using big data analysis with deep learning in anomaly detection shows excellent combination that may be optimal solution as deep learning needs millions of samples in dataset and that what big data handle and what we need to construct big model of normal behavior that reduce false-

positive rate to be better than small traditional anomaly models.

In this paper, we have presented an anomaly based intrusion detection system for IoT security with the performance comparison of different learning algorithms and methods in semi-supervised way. Based on our results, we see that all the algorithms achieved pretty much good results except Transformer in different threshold. In the future, more datasets from different environment should be tested in the ML/DL methods used in this study. This can help to further clarify the performance, time cost and comparison between the methods.

References:

[1] Ericsson. Ericsson Mobility Report November 2019. Nov. 2019. url: https://www.ericsson.com/4acd7e/assets/local/mobilityreport/documents/2019/emr-november-2019.pdf.

[2] Security Aspects of 5G for Industrial Networks. 2020. url: https://www.5gacia. org / publications / security-aspects-of-5g-for industrial -networks/.

[3] Monowar Bhuyan, Dhruba K Bhattacharyya, and Jugal Kalita. Network Traffic Anomaly Detection and Prevention: Concepts, Techniques, and Tools. Jan.2017. isbn: 978-3-319-65186-6. doi: 10.1007/978-3-319-65188-0.

[4] A. Sperotto et al. "An Overview of IP Flow-Based Intrusion Detection". In:IEEE Communications Surveys Tutorials 12.3 (2010), pp. 343–356.

[5] R. Sommer and V. Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection". In: 2010 IEEE Symposium on Security and Privacy. May 2010, pp. 305–316. doi: 10.1109/SP.2010.25.

[6] Markus Ring et al. "A Survey of Network-based Intrusion Detection Data Sets". In: CoRR abs/1903.02460 (2019). arXiv: 1903.02460. url: http://arxiv.org/abs/1903.02460.

[7] Sebastián García et al. "An Empirical Comparison of Botnet Detection Methods". In: Computers & Security 45 (Sept. 2014), pp. 100–123. doi:10.1016/j.cose.2014.05.011.

[8] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: Jan. 2018, pp. 108–116. doi: 10.5220/0006639801080116.

[9] Maria Jose Erquiaga Agustin Parmisano Sebastian Garcia. Stratosphere Laboratory. A labeled dataset with malicious and benign IoT network traffic. Jan. 2020. url: https://www.stratosphereips.org/datasets-iot23.

[10] Gabriel Maciá-Fernández et al. "UGR'16: A new dataset for the evaluation of

cyclostationarity-based network IDSs". In: Computers & Security 73 (Nov.2017). doi: 10.1016/j.cose.2017.11.004.

[11] Raghavendra Chalapathy and Sanjay Chawla. "Deep Learning for Anomaly Detection: A Survey". In: CoRR abs/1901.03407 (2019). arXiv: 1901.03407.url: http://arxiv.org/abs/1901.03407.

[12] Vít Skvára, Tomás Pevný, and Václav Smídl. "Are generative deep models for novelty detection truly better?" In: CoRR abs/1807.05027 (2018). arXiv:1807.05027. url: http://arxiv.org/abs/1807.05027.

[13] Bo Zong et al. "Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection". In: International Conference on Learning Representations.2018. url: https://openreview.net/forum?id=BJJLHbb0-.

[14] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. 2013.arXiv: 1312.6114 [stat.ML].

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. http://www.deeplearningbook.org. MIT Press, 2016.

[16] Mahbod Tavallaee et al. "A detailed analysis of the KDD CUP 99 data set". In: IEEE Symposium. Computational Intelligence for Security and Defense Applications, CISDA 2 (July 2009). doi: 10.1109/CISDA.2009.5356528.

[17] Mikhail Mishin. "Anomaly Detection Algorithms and Techniques for Network Intrusion Detection Systems". In: (June. 2020).

[18] Christopher Kruegel and Giovanni Vigna. "Anomaly Detection of Web-Based Attacks". In: Proceedings of the 10th ACM Conference on Computer and Communications Security. CCS '03. Washington D.C., USA: Association for Computing Machinery, 2003, pp. 251–261. isbn: 1581137389. doi: 10.1145/948109.948144. url: https://doi.org/10.1145/948109.948144.

[19] Matthew V. Mahoney and Philip K. Chan. “Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks”. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD ’02. Edmonton, Alberta, Canada: Association for Computing Machinery, 2002, pp. 376–385. isbn: 158113567X. doi: 10.1145/775047.775102. url: https://doi.org/10.1145/775047.775102.

[20] Federico Simmross-Wattenberg et al. “Anomaly Detection in Network Traffic Based on Statistical Inference and alpha-Stable Modeling”. In: Dependable and Secure Computing, IEEE Transactions on 8 (July 2011), pp. 494–509. doi:10.1109/TDSC.2011.14.

[21] Huy Anh Nguyen et al. “Network traffic anomalies detection and identification with flow monitoring”. In: 2008 5th IFIP International Conference on Wireless and Optical Communications Networks (WOCN ’08). May 2008, pp. 1–5. doi:10.1109/WOCN.2008.4542524.

[22] Eleazar Eskin et al. “A Geometric Framework for Unsupervised Anomaly Detection”. In: Applications of Data Mining in Computer Security. Ed. by Daniel Barbará and Sushil Jajodia. Boston, MA: Springer US, 2002, pp. 77–101. isbn: 978-1-4615-0953-0. doi: 10.1007/978-1-4615-0953-0_4. url:https://doi.org/10.1007/978-1-4615-0953-0_4.

[23] Haakon Ringberg et al. “Sensitivity of PCA for traffic anomaly detection”. In:vol. 35. June 2007, pp. 109–120. doi: 10.1145/1254882.1254895.

[24] Daniela Brauckhoff, Kave Salamatian, and Martin May. “Applying PCA for Traffic Anomaly Detection: Problems and Solutions”. In: May 2009, pp. 2866–2870. doi: 10.1109/INFCOM.2009.5062248.

[25] Yoshiki Kanda et al. “ADMIRE: Anomaly detection method using entropybased PCA with three-step sketches”. In: Computer Communications 36.5(2013), pp. 575–588. issn: 0140-3664.

doi: https://doi.org/10.1016/j.comcom.2012.12.002. url:
http://www.sciencedirect.com/science/article/pii/S0140366412003994.

[26] Nour Moustafa and Jill Slay. “UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)”. In:Nov. 2015. doi: 10.1109/MilCIS.2015.7348942.

[27] Leyla Bilge et al. “Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis”. In: Dec. 2012, pp. 129–138. doi:10.1145/2420950.2420969.

[28] Donghwoon Kwon et al. “An Empirical Study on Network Anomaly Detection Using Convolutional Neural Networks”. In: July 2018, pp. 1595–1598. doi:10.1109/ICDCS.2018.00178.

[29] Pablo Torres et al. “An Analysis of Recurrent Neural Networks for Botnet Behavior Detection”. In: June 2016. doi: 10.1109/ARGENCON.2016.7585247.

[30] N. Shone et al. “A Deep Learning Approach to Network Intrusion Detection”. In: IEEE Transactions on Emerging Topics in Computational Intelligence 2.1(2018), pp. 41–50.

[31] Pierre Baldi. “Autoencoders, Unsupervised Learning, and Deep Architectures”. In: Proceedings of ICML Workshop on Unsupervised and Transfer Learning. Ed. by Isabelle Guyon et al. Vol. 27. Proceedings of Machine Learning Research. Bellevue, Washington, USA: PMLR, Feb. 2012, pp. 37–49. url:http://proceedings.mlr.press/v27/baldi12a.html.

[32] Yang Yu, Jun Long, and Zhiping Cai. “Network Intrusion Detection through Stacking Dilated Convolutional Autoencoders”. In: Security and Communication Networks 2017 (Nov. 2017), pp. 1–10. doi: 10.1155/2017/4184196.

[33] Yisroel Mirsky et al. “Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection”. In: Jan. 2018. doi: 10.14722/ndss.2018.23211.

[34] Jinwon An and Sungzoon Cho. “Variational Autoencoder based Anomaly Detection using Reconstruction Probability”. In: 2015.

[35] Haowen Xu et al. “Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications”. In: CoRR abs/1802.03903(2018). arXiv: 1802.03903. url: http://arxiv.org/abs/1802.03903.

[36] Manuel Lopez-Martin et al. “Conditional Variational Autoencoder for Prediction and Feature Recovery Applied to Intrusion Detection in IoT”. In: Sensors 17 (Aug. 2017), p. 1967. doi: 10.3390/s17091967.

[37] Filipe Falcão et al. “Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection”. In: Apr. 2019, pp. 318–327. doi:10.1145/3297280.3297314.

[38] James F. Kurose and Keith W. Ross. Computer Networking: A Top-Down Approach (6th Edition). 6th. Pearson, 2012. isbn: 0132856204.

[39] Achiv Chauha and Palak Jain. TCP/IP Model. 2020. url: https://www.geeksforgeeks.org/tcp-ip-model/.

[40] Science Commerce. “A "Kill Chain" analysis of the 2013 target data breach”.In: Jan. 2014, pp. 41–60.

[41] Cloudflare Learning Center. 2020. url: https://www.cloudflare.com/learning/.

[42] Mirai (malware). 2020. url: https://en.wikipedia.org/wiki/Mirai_(malware).

[43] Khloud Al Jallad1 , Mohamad Aljnidi and Mohammad Said Desouki. "Anomaly detection optimization using big data and deep learning to reduce false-positive".(2020) 7:68

[44] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly Detection: A Survey”. In: ACM Comput. Surv. 41.3 (July 2009). issn: 0360-0300. doi:10.1145/1541880.1541882. url: https://doi.org/10.1145/1541880.1541882.

[45] Ayyoob Hamza et al. “Clear as MUD: Generating, Validating and Applying IoT Behaviorial Profiles (Technical Report)”. In: CoRR abs/1804.04358 (2018).arXiv: 1804.04358. url: http://arxiv.org/abs/1804.04358.

[46] B Claise, Brian Trammell, and P Aitken. “Specification of the IP Flow Information Export

(IPFIX) Protocol for the Exchange of Flow Information".In: (Sept. 2013).

[47] Yue Liang, Nikhil Vankayalapati, "Machine Learning and Deep Learning Methods for Better Anomaly Detection in IoT-23 Dataset Cybersecurity".

[48] Mishin, Mikhail. "Anomaly Detection Algorithms and Techniques for Network Intrusion Detection Systems." (2020).

[49] D. D. Monda, G. Bovenzi, A. Montieri, V. Persico and A. Pescapè, "IoT Botnet-Traffic Classification Using Few-Shot Learning," 2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy, 2023, pp. 3284-3293, doi: 10.1109/BigData59044.2023.10386602.

[50] Sebastian Garcia, Agustin Parmisano, & Maria Jose Erquiaga. (2020). IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0) [Data set]. Zenodo. http://doi.org/10.5281/zenodo.4743746.

[51] Barut, O., Luo, Y., Zhang, T., Li, W., and Li, P. (2020). Netml: A challenge for network traffic analytics. arXiv preprint arXiv:2004.13006.

[52] Dutta, V., Chora´s, M., Pawlicki, M., and Kozik, R. (2020b). Hybrid model for improving the classification effectiveness of network intrusion detection. In Conference on Complex, Intelligent, and Software Intensive Systems.Springer.