2024

Optimizing Neural Networks for IIoT Attack Detection

Hamad Alblooshi
hha7620@rit.edu

Rochester Institute of Technology

Optimizing Neural Networks for IIoT Attack Detection

By

Hamad Alblooshi

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree in

Master of Science in Cybersecurity

Department of Computing Sciences

Supervised by

Dr. Kevser Ovaz Akpinar

Department of Electrical Engineering and Computing Sciences

Rochester Institute of Technology

Dubai

2024

Acknowledgments

I would like to express my deepest appreciation and gratitude to Dr. Kevser Ovaz Akpinar for her endless support and assistance throughout my journey in completing this thesis. Her guidance, insightful feedback, constructive criticism, and encouragement have been instrumental in shaping this work, which has significantly improved the quality of this thesis. Moreover, to extend my appreciation and gratitude to Dr. Wesam Almobaideen, Dr. Ali Assi, and Dr. Huda Saadeh for their invaluable feedback and insightful suggestions, which profoundly enriched this research and were crucial to its success.

Abstract

The Industrial Internet of Things (IIoT) stands as a revolutionary force, intertwining physical devices, sensors, and industrial systems to usher in advanced automation and data-driven decision-making across various sectors. However, this increased connectivity has exposed these systems to a growing array of cyber threats. Safeguarding IIoT environments becomes crucial to maintain the integrity, availability, and confidentiality of critical industrial processes. In response, this research explores the optimization of neural network parameters using Genetic Algorithms (GA). The application of GA has led to achieve a remarkable accuracy of 95% across various attack types. The results demonstrate a high performance in identifying complex attack patterns, contributing to the resilience of IIoT systems against emerging cyber threats.

Keywords: Industrial Internet of Things (IIoT), Artificial Intelligence (AI), machine learning (ML), Genetic Algorithm (GA), Neural Network (NN)

Table of Contents

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviations	Definition
ABC	Artificial Bee Colony Algorithm
AI	Artificial Intelligence
BLR	Binomial Logistic Regression
CNN	Convolutional Neural Network
DBF	Deep Belief Network
DDoS	Distributed Denial-of-Service
DNN	Deep Neural Network
DT	Decision Tree
FL	Federated Learning
GA	Genetic Algorithm
GB	Gradient Boosting
GNB	Gaussian Naive Bayes
HIDS	Host Intrusion Detection System
ICS	Industrial Control Systems
IDPS	Intrusion Detection and Prevention Systems
IDS	Intrusion Detection System
IDS	Intrusion Detection System
IIoT	Industrial Internet of Things
Industry 4.0 or 4IR	Fourth Industrial Revolution
IOC	Indicators of Compromise
IoT	Internet of Things
IT	Information Technology
KNN	K-Nearest Neighbours
LSTM	Long Short-Term Memory
MEC	Mobile Edge Computing
NB	Naive Bayes
NIA	Nature-Inspired Algorithms
NIDS	Network Intrusion Detection System

NN	Neural Network
OT	Operational Technology
ReLU	Rectified Linear Unit
RF	Random forest
RNN	Recurrent Neural Network
SDN	Software-Defined Networking
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TL	Transfer Learning
TTP	Tactics, Techniques, and Procedures

1- Introduction

In recent years, the integration of digital technologies into industrial processes has fundamentally transformed the way we perceive and manage industrial systems [1]. This convergence of physical machinery with digital infrastructure, commonly referred to as the IIoT, holds the promise of unprecedented levels of efficiency, productivity, and connectivity across various industrial sectors [2]. However, alongside these advancements comes the looming threat of cyber-attacks targeting critical infrastructure and industrial control systems [3].

As these systems become more interconnected and central to industrial operations, they also become attractive targets for sophisticated cyber threats [4]. The complexity and scale of these threats underscore the urgent need for robust detection mechanisms capable of identifying and mitigating potential attacks. Traditional cybersecurity approaches often fall short in effectively addressing the dynamic and evolving nature of cyber-attacks within IIoT environments, struggling to keep pace with the increasingly sophisticated tactics employed by malicious actors.

This situation calls for innovative strategies that can not only adapt to new threats but also respond swiftly and effectively. Beyond the technical realm, the implications of securing IIoT systems extend into broader societal and economic dimensions. Safeguarding industrial infrastructures from cyber threats is not merely a technical challenge but a critical necessity for ensuring operational continuity, protecting sensitive data, and maintaining public safety [5].

This thesis addresses the pressing need for advanced cybersecurity solutions tailored to the unique challenges of IIoT environments. Specifically, it proposes the development of an advanced machine-learning algorithm designed to detect cyber-attacks targeting industrial systems. By leveraging the capabilities of machine learning and utilizing the rich data inherent in IIoT networks, this approach aims to enhance the security posture of industrial organizations and reduce the risks posed by cyber threats.

1.1. Background

Before delving into the complexities of the IIoT, machine learning (ML), and Intrusion Detection Systems (IDS), it's crucial to establish a foundational understanding of the interaction between industrial systems and cybersecurity. Industrial environments or operational technology (OT), characterized by the integration of physical machinery with digital infrastructure, have undergone a deep transformation with the advent of IIoT technologies [6]. While IIoT promises enhanced efficiency and connectivity, it also introduces new challenges in terms of cybersecurity [7]. As industrial systems become increasingly interconnected and reliant on digital technologies, they become more susceptible to cyber threats targeting critical infrastructure and control systems. This requires the development of advanced cybersecurity solutions tailored specifically to the unique challenges posed by IIoT environments. In the subsequent sections, the focus will shift to exploring the fundamental concepts of the IIoT, ML, and IDS, highlighting their roles in supporting cybersecurity.

1.1.1. Industrial Internet of Things (IIoT)

The Industrial Internet of Things (IIoT) refers to the interconnected network of devices, sensors, and systems that are used in industrial and operational environments. IIoT can significantly help in collecting, exchanging, and analyzing data to optimize processes, improve efficiencies, and enable remote monitoring and control [8]. The adoption of IIoT has grown rapidly in recent years, transforming industries such as manufacturing, logistics, energy, and transportation into what is known as the Fourth Industrial Revolution (Industry 4.0 or 4IR).

Industry 4.0 refers to the fourth industrial revolution; it is the current and ongoing transformation of industries and societies through the integration of advanced digital technologies. It is characterized by the convergence of physical, digital, and biological technologies, blurring the lines between the physical and virtual worlds [9]. At the heart of industry 4.0 are technologies such as artificial intelligence (AI), big data, the IoT which the IIoT is derived from, cloud computing, robotics, and other emerging technologies.

These technologies are changing the way businesses operate, driving innovation, and creating new opportunities. As an example, in manufacturing, the use of smart factories powered by IoT and AI enables autonomous and connected production processes, leading to

increased efficiency, reduced costs, and improved quality control. However, along with the numerous benefits of IIoT, there has been a parallel rise in cybersecurity threats that pose significant risks to the secure and reliable operation of IIoT systems.

As IIoT systems become more interconnected and data-driven, they are vulnerable to various cyber threats, including unauthorized access, data breaches, malware attacks, and insider threats. These threats can result in the disruption of operations, financial losses, damage to reputation, and potential safety hazards. Furthermore, the increasing convergence of IT (Information Technology) and OT in IIoT systems has created new attack vectors and complexities in securing these systems. The growing reliance on IIoT in critical infrastructure, such as power grids, transportation networks, and industrial control systems (ICS), has raised concerns about the potential for cyber-attacks to have far-reaching and devastating consequences. Cybercriminals, nation-state actors, and other malicious entities are constantly evolving their tactics, techniques, and procedures (TTPs) to exploit vulnerabilities in IIoT systems, making it imperative for organizations to prioritize robust cybersecurity measures to safeguard their IIoT deployments [10].

Considering these challenges, ensuring the security of IIoT systems has become a critical priority for industries and organizations that leverage IIoT technologies. This includes implementing strong authentication and access controls, securing device firmware and software, encrypting data, implementing intrusion detection and prevention systems (IDPS), and having robust incident response plans in place. Furthermore, compliance with relevant regulations and industry standards, as well as regular security audits, vulnerability assessments, and penetration testing, are essential components of a comprehensive IIoT cybersecurity strategy. While the adoption of IIoT offers significant benefits for industrial and operational environments, the growth of cybersecurity threats poses significant challenges. Organizations must prioritize robust cybersecurity measures to safeguard their IIoT systems and protect against evolving threats. With the increasing reliance on IIoT in critical infrastructure and industrial processes, the importance of IIoT cybersecurity cannot be overstated in ensuring the secure and reliable operation of IIoT systems.

1.1.2. Machine Learning

Machine learning is a subset of AI that involves the use of algorithms and statistical models to enable computer systems to learn from and make predictions or decisions based on the given data [11]. In the realm of cybersecurity, machine learning can be a powerful tool to detect, prevent, and respond to cyber threats more proactively and efficiently.

Machine learning algorithms can be trained on large datasets of cybersecurity-related data, such as network traffic, log files, malware samples, and user behaviour to learn patterns, anomalies, and indicators of compromise (IOCs). Once trained, these models can be deployed in real-time to analyze incoming data and identify potential cyber threats, such as malware, viruses, phishing attacks, and intrusions. One of the key applications of machine learning in cybersecurity is threat detection [12]. For example, machine learning can be used to identify suspicious patterns of network traffic, abnormal user behaviour, or unknown malware samples. These models can also adapt and learn from new data, allowing them to evolve and improve their detection capabilities over time [13].

Another application of machine learning in cybersecurity is in vulnerability assessment and patch management. Machine learning models can analyze system configurations, patch histories, and other data to identify vulnerabilities in software or hardware that could be exploited by cybercriminals [14]. This information can help organizations prioritize their patch management efforts and proactively address potential vulnerabilities before they are exploited.

Machine learning techniques can also be used in cybersecurity for threat hunting and incident response. By analyzing historical data and patterns of cyber-attacks, machine learning models can help identify ongoing or potential cyber threats that may have evaded traditional security measures. This can aid in proactive threat-hunting efforts, allowing organizations to detect and respond to threats in a more timely and effective manner [15].

Moreover, machine learning can be used to enhance authentication and access control mechanisms in cybersecurity. Machine learning models can analyze user behaviour patterns,

device characteristics, and other contextual information to detect anomalies or suspicious activities that may indicate unauthorized access or compromised accounts. This can help organizations detect and prevent unauthorized access or insider threats, enhancing the security of their systems and data [16].

However, it's important to note that machine learning in cybersecurity is not the ultimate solution and it has limitations. Machine learning models can produce false positives or false negatives and can also be susceptible to adversarial attacks. Therefore, it's crucial to continuously evaluate, validate, and update machine learning models to ensure their accuracy and effectiveness in the ever-evolving landscape of cybersecurity threats [17]. Machine learning has the potential to significantly enhance cybersecurity by enabling proactive threat detection, vulnerability assessment, incident response, and access control. By leveraging the power of data and algorithms, machine learning can help organizations stay ahead of cyber threats and better protect their systems, networks, and data from cyber-attacks [18].

1.1.3. Intrusion Detection Systems

Intrusion Detection Systems (IDS), including both Network-based IDS (NIDS) and Host-based IDS (HIDS), serve as vital components in the complex landscape of cybersecurity, acting as guardians against potential threats to network and system integrity [19]. These systems are meticulously crafted to address the dynamic and ever-evolving nature of cyber threats. Their overarching objective is to provide organizations with an intelligent and automated layer of defence, capable of detecting and responding to a multitude of security incidents [20].

NIDS is a security system that functions as a sentinel overseeing the collective traffic coursing through interconnected systems. Employing advanced algorithms, it scrutinizes packets of data traversing the network in real-time. By comparing observed patterns against predefined signatures indicative of known threats, NIDS detects activities such as port scans, denial-of-service attacks, and other malicious behaviours that target network vulnerabilities. Its ability to analyze the broader network landscape ensures a comprehensive defence against threats targeting the organization's interconnected infrastructure [21].

HIDS focus on the individual hosts or devices within a network. It monitors activities on each host, including file integrity, system logs, and system calls. This approach allows HIDS to detect anomalies specific to the activities on a particular host, such as unauthorized access attempts, unusual file modifications, or suspicious processes. HIDS provides a localized perspective, complementing the broader network awareness provided by NIDS [22].

Both NIDS and HIDS operate by leveraging sophisticated algorithms and rule sets. These algorithms are meticulously designed to recognize subtle patterns and anomalies within the vast array of network traffic or host activities. By comparing observed data against established signatures and behavioural baselines, both NIDS and HIDS can discern activities indicative of malicious intent, including deviations from security policies and potential policy violations. Furthermore, NIDS and HIDS play pivotal roles in identifying and mitigating security policy violations. By scrutinizing activities against predefined rules and policies, they ensure that organizations adhere to their established security postures [23]. This capability is particularly crucial in maintaining compliance with industry regulations and safeguarding sensitive data from unauthorized access or manipulation. The adaptability of NIDS and HIDS is another key feature. The detection systems are continually upgraded to keep pace with the evolving of cyber threats. They constantly update their knowledge bases, incorporating new ads and refining detection mechanisms [24]. This adaptive quality enables NIDS and HIDS to stay ahead of emerging threats, making them indispensable assets in an organization's cybersecurity arsenal.

Key features of an IDS are as follows [25], [26], [27], [28], [29], [30], [31], [32]:

1. **Signature-based Detection:** Signature-based detection is a fundamental technique employed by IDS, involving the comparison of observed data against predefined signatures or patterns of known cyber threats. These signatures are essentially fingerprints of malicious activities, allowing the IDS to recognize and respond to well-documented attacks, such as viruses, worms, and specific intrusion methods.
2. **Behaviour-based Detection:** Behaviour-based detection takes a more dynamic approach by establishing a baseline of normal behaviour for networks or hosts. The IDS continuously monitors activities and flags deviations from this established baseline as potential security incidents. This method is particularly effective in detecting

previously unknown or evolving threats that might not be covered by signature-based approaches.

3. **Rule-based Detection:** Rule-based detection involves the application of rules and algorithms to identify potential threats based on general characteristics of attacks. This method allows IDS to adapt to emerging threats by leveraging behavioural analysis and situational awareness. Rule detection is especially valuable in scenarios where rigid signatures may not capture the full spectrum of attack variations.
4. **Real-time Monitoring:** One of the defining features of IDS is its real-time monitoring capability. This enables the system to promptly detect and respond to security incidents as they unfold. The immediate identification of malicious activities is crucial in minimizing the potential impact of cyber threats and fortifying the overall security posture of a network or system.
5. **Alerts and Notifications:** When an IDS identifies suspicious behaviour, it generates alerts or notifications to prompt further investigation or action. These alerts provide valuable insights into the nature of the detected incident, allowing cybersecurity professionals to assess the severity of the threat and implement appropriate response measures.
6. **Logs and Reporting:** IDS systems maintain detailed logs of activities, offering a comprehensive record of events for retrospective analysis. These logs not only aid in understanding the specifics of security incidents but also serve as valuable resources for fine-tuning the IDS, refining detection rules, and improving overall security strategies.

1.2. Problem Statement

The rapid adoption of the IIoT has led to a new era of industrial automation and efficiency, revolutionizing the way operations are conducted. This transformation, fuelled by increased connectivity and the seamless integration of digital technologies, has undoubtedly enhanced productivity. However, this very connectivity exposes critical infrastructures to an expanding array of sophisticated cyber threats, raising significant concerns about the robustness of cybersecurity measures in IIoT systems [33]. The necessity of ensuring the

cybersecurity of IIoT systems is crucial, as any compromise in the integrity and reliability of industrial operations could have far-reaching consequences [34].

The convergence of OT and IT in industrial environments creates complex attack surfaces, vulnerable to a diverse range of cyber threats. From Distributed Denial of Service (DDoS) attacks to data breaches and malware infiltration, the threat landscape confronting industrial systems is multifaceted and constantly evolving. Incidents like the Stuxnet and Triton attacks serve as stark reminders of the potential consequences of compromised industrial security, with the ability to disrupt critical infrastructure and endanger lives [35].

The integrity and reliability of industrial operations depend on the security of IIoT systems. Any compromise in cybersecurity could lead to disruptions in production processes, financial losses, reputational damage, and, in extreme cases, pose risks to human safety [36]. Thus, ensuring the resilience of IIoT systems against cyber threats is not only a matter of operational continuity but also a crucial aspect of safeguarding critical infrastructure and maintaining public trust.

2- Literature Review

2.1. Related work

This section examines a series of research papers contributing to the growing landscape of machine learning applications within IIoT environment. Each of these studies employs various methodologies and models to develop robust intrusion detection systems tailored for IIoT environments. The cornerstone of these investigations lies in the utilization of a variety of algorithms tailored specifically for IIoT IDS, which serve as essential elements for algorithm development and evaluation. This shows the importance of using tailored algorithms designed for IIoT IDS, providing insights into various techniques used to strengthen the detection capabilities to secure the industrial environment from any potential threats or cyberattacks. Through an analysis of performance metrics such as Accuracy, Precision, F1 Score, and Recall, this section provides insights into the effectiveness of machine learning algorithms and

frameworks, from leveraging IIoT datasets, thereby enhancing the security posture of IIoT infrastructures.

In [37] authors introduce FMDADM, a multi-layer framework specifically designed to detect and mitigate DDoS attacks in SDN-based IoT networks using machine learning techniques. The framework operates across different layers, including the data plane, control plane, and application plane, to provide comprehensive defence against DDoS attacks. FMDADM leverages machine learning algorithms, including SVM, GNB, KNN, BLR, DT, and RF, for traffic classification and anomaly detection, enabling accurate identification and differentiation between normal and malicious network behaviour. The study utilizes the Edge-IIoTset dataset, focusing specifically on DDoS attacks for evaluation purposes, achieving performance metrics with an accuracy of 99.79%, precision of 99.09%, F1 score of 99.43%, and recall of 99.77%. Furthermore, the paper outlines the parameters used for model optimization, including the splitting criterion, number of trees, minimum samples per leaf, minimum samples per split, and maximum features, with selected values of Gini, 1000, 1, 2, and 7.

The [38] authors introduces an architecture based on Mobile Edge Computing (MEC) to enhance the security of IoT applications using federated deep learning. The proposed architecture aims to address the security challenges in IoT systems by leveraging the computational capabilities of MEC servers and the privacy-preserving nature of federated learning. By distributing the deep learning model training process across multiple edge servers, the architecture ensures that sensitive data remains on the local edge servers, thus preserving privacy. The federated deep learning approach allows for collaborative model training without the need for centralized data collection, minimizing the risk of data breaches. The research achieves notable results with an accuracy of 86%, precision of 95%, F1 score of 87%, and recall of 87% utilizing the NSL-KDDTest dataset.

In [39] authors explore machine learning approaches for detecting and characterizing cyber-attacks in IoT-enabled cyber-physical systems. It highlights the growing threat landscape and the need for effective defence mechanisms in the context of IoT systems. It discusses the

challenges and considerations specific to OT environments, including limited resources and the need for real-time analysis. The paper used Deep Neural Networks (DNN) combined with the Decision Trees algorithm. The proposed solution was tested in an OT environment by using the natural gas pipeline dataset. The research achieves notable results with an accuracy of 97.3%, precision of 97%, F1 score of 90%, and recall of 97%.

In [40] authors present an AI-enabled framework designed for detecting distributed cyber-attacks in IoT-based smart environments. The proposed framework leverages DNN, SVM RF, DT, GB, and NB, to analyze network traffic patterns and identify malicious activities. By deploying distributed AI models across IoT devices and gateways, the framework enables real-time detection and response to cyber-attacks, while minimizing resource consumption and latency. It was conducted on the Edge-IIoTset dataset achieving an accuracy of 96%, precision of 86%, F1 score of 85%, and recall of 85%.

The [41] authors present a hybrid solution that combines a CNN and LSTM model for intrusion detection at the edge of the IIoT systems. The proposed model combines the strengths of CNN for spatial feature extraction and LSTM for capturing temporal dependencies in the third and fourth network layers. It takes into consideration privacy concerns by employing a privacy-aware feature selection algorithm that removes sensitive data before analysis. Notably, the study utilizes the IoT-DS2 dataset, which is constructed by combining different datasets, including BoT-IoT, IoT-NI, MQTT-IoTIDS2020, MQTTset, and IoT-23, rebuilt from PCAP files. The results demonstrate achieving an accuracy of 97.14%, precision of 82.32%, F1 score of 72.66%, and recall of 74.62%.

In [42] the authors propose an approach to enhance cybersecurity in the context of the IIoT. The authors present a hybrid deep learning-based intrusion detection system specifically tailored for IIoT environments. By leveraging the power of convolutional neural networks (CNN) and long short-term memory (LSTM), the proposed system effectively identifies and classifies intrusions in real-time. The hybrid nature of the system combines the strengths of both CNN LSTM models to improve detection accuracy and minimize false positives and false negatives. Additionally, it provides valuable insights into the hyperparameters such as 100 epochs, a batch size of 5000, a learning rate of 0.0001, activation functions including Relu and

Sigmoid, spatial dropout regularization, and MinMaxScaler normalization which ended up with an accuracy of 98.69%.

The [43] authors discuss intrusion detection in the context of IIoT systems. It addresses the unique challenges and requirements for intrusion detection in industrial environments, where critical infrastructure and OT are interconnected. The paper proposed two methods using deep learning to classify IIoT network packets and highlights the importance of real-time monitoring and response to mitigate potential threats. It emphasizes the need for a comprehensive approach that combines network monitoring, behavioural analysis, and machine learning algorithms to effectively detect and respond to intrusions in IIoT systems, ultimately enhancing the security and resilience of industrial environments. The proposed RNN (Recurrent Neural Network) architecture employs three hidden layers for binary classification and four for multi-class. It utilizes 90 hidden nodes for binary and 120 for multi-class classification, with a consistent learning rate of 0.1. for both. Relu is the activation function for binary, while sigmoid is used for multi-class. Both classifications utilize softmax for output interpretation, and the Adam optimizer, achieving an accuracy of 99%.

In [44] the authors focus on intrusion detection for IoT applications using federated and transfer learning techniques. It highlights the challenges of traditional intrusion detection systems in IoT due to limited resources and the distributed nature of IoT devices. The paper proposes a federated learning approach where local models are trained on individual IoT devices, and then a global model is created by aggregating the knowledge from these local models while preserving data privacy. Additionally, transfer learning is employed to adapt the global model to new IoT environments with limited labeled data. The study leverage from the Edge-IIoTset dataset achieves an accuracy of 98.1%, precision of 99%, F1 score of 99%, and recall of 99%.

The [45] authors introduce a one-class classifier based on polynomial interpolation and apply it to networking security. The proposed classifier aims to detect anomalies or malicious activities in network traffic by learning the normal behaviour of the system using a polynomial interpolation model. By capturing the underlying patterns and trends in the data, the classifier can distinguish between normal and abnormal network behaviour. Leveraging the Edge-

IIoTset dataset, the classifier achieves an accuracy of 97.27%, precision of 96.03%, F1 score of not given, and recall of 94.82%. Additionally, PCA (Principal Component Analysis) is performed with a value of 15.

In [46] the authors present a security model based on LightGBM (Gradient Boosting) and transformer architectures to safeguard healthcare systems against cyberattacks and highlight the criticality of healthcare systems and the increasing risk of cyber threats. LightGBM is employed for anomaly detection, effectively identifying malicious activities within the healthcare system. The Transformer architecture, known for its proficiency in sequential data analysis, is leveraged for log analysis and generating predictions. Experimental results demonstrate the model achieved a precision of 92%, F1 score of 89%, and a recall of 88%, although the accuracy is not clear in the paper.

Table 1: Related Work

Paper	Proposed solution	Application/ Domain	Dataset	Traffic types	Accuracy	Precision	F1	Recall
[34]	FMDADM	SDN-based IoT networks	Edge-IIoTset	1	99.79	99.09	99.43	99.77
[35]	Federated Learning Architecture	IoT Applications	NSL-KDDTest	Not Given	86	95	87	87
[36]	DNN and Decision Trees	IoT-enabled cyber-physical systems	Natural Gas Pipeline Dataset	Not Given	97.3	97	90	97
[37]	Distributed AI Framework	IoT-based smart environments	Edge-IIoTset	15	96	86	85	85
[38]	Hybrid CNN-LSTM Model	IIoT Systems at the edge	IoT-DS2	23	97.14	82.32	72.66	74.62
[39]	Hybrid Deep Learning Intrusion Detection	IIoT Environments	Edge-IIoTset	15	98.69	Not Given	Not Given	Not Given
[40]	Deep Learning for IIoT Intrusion Detection	IIoT Systems	Edge-IIoTset	15	99	Not Given	Not Given	Not Given

[41]	Federated and Transfer Learning	IoT Applications	Edge-IIoTset	15	98.1	99	99	99
[42]	Polynomial Interpolation One-Class Classifier	Networking Security	Edge-IIoTset	15	97.27	96.03	Not Given	94.82
[43]	LightGBM and Transformer Security Model	Healthcare Systems	Edge-IIoTset	15	Not Given	92	89	88

Table 1 summarizes various studies that support the importance of developing different techniques and methods to confront cyber-attacks targeting IIoT environments. Notably, Edge-IIoTset emerges as a prevalent dataset across multiple studies, indicating its significance as a foundational resource for evaluating machine learning solutions. High accuracy rates have been observed in these studies, with most achieving above 95%, validating the efficacy of the proposed methodologies in detecting and mitigating industrial cyber threats. FMDADM solution was able to achieve a remarkable accuracy of 99.79%, focusing exclusively on DDoS attacks in SDN-based IoT networks. Furthermore, the diversity of applications, ranging from healthcare systems to IoT-enabled cyber-physical systems, showcases the broad applicability of these machine-learning solutions across various IIoT domains.

2.2. Motivation

This study is motivated by the wish to acquire and implement innovative techniques that can greatly improve cybersecurity in critical sectors. The increasing complexity and interconnection of industrial systems call for creative strategies tailored to tackle the distinct challenges presented by the IIoT. This research seeks to safeguard critical infrastructure by utilizing cutting-edge methods and promoting the security and dependability of industrial operations.

Another reason is the challenge of developing successful approaches for a complicated and ever-changing setting such as IIoT, where standard cybersecurity techniques frequently prove to be ineligible. The complex design of these systems, along with the changing environment of cyber threats, requires new tactics that can adjust and detect to possible risks immediately. Addressing these challenges offers a great chance to have a significant protection layer for the security of critical industrial systems.

3- Methodology

3.1. Dataset

In the initial stages of this research, different datasets have been explored, focusing on datasets about cyber attacks within the IIoT domain. Among the array of datasets considered, these datasets have been reviewed in this thesis: N-BaIoT, Bot-IoT, MQTTset, FederatedTon_IoT, X-IIoTD, WUSTL-IIoT-2021, and Edge-IIoTset. After the evaluation as shown below in Table 2, the Edge-IIoTset dataset emerged as the preferred choice, primarily due to its comprehensive coverage across all the seven layers of the IoT architecture [47] which are given later in this section, offers a rich feature set of 61 attributes, and includes 14 attack types along with normal traffic. Furthermore, its status as one of the latest datasets to the IIoT landscape, along with its widespread adoption can be used as a benchmark in this research.

Table 2: IoT IDS Datasets

Ref	Dataset	Year	Features	Layers	Devices	Attacks	IIoT
[48]	N-BaIoT	2018	23	2	9	10	No
[49]	Bot-IoT	2019	46	VM	Simulation	8	No
[50]	MQTTset	2020	33	2	8	5	No
[51]	FederatedTon_IoT	2020	31	3	Simulation	9	No
[52]	X-IIoTD	2021	59	3	Not Given	18	Yes
[53]	WUSTL-IIOT-2021	2021	41	4	5	4	Yes
[54]	Edge-IIoTset	2022	61	7	10	14	Yes

The Edge-IIoTset dataset [54] was designed to simulate realistic cyber security scenarios on the IoT and IIoT applications. The dataset is intended to be used for training and evaluating machine learning models for cyber security in centralized and federated learning settings. The architecture of Edge-IIoTset includes multiple layers: Cloud Computing Layer, Network Functions Virtualization Layer, Blockchain Network Layer, Fog Computing Layer, Software-Defined Networking Layer, Edge Computing Layer, and IoT and IIoT Perception Layer.

Table 3: Edge-IIoTset Size

N	Traffic Type	Count	Percentage
1	Normal	24301	15.40%
2	DDoS_UDP	14498	9.19%
3	DDoS_ICMP	14090	8.93%
4	Ransomware	10925	6.92%
5	DDoS_HTTP	10561	6.69%
6	SQL_injection	10311	6.53%
7	Uploading	10269	6.51%
8	DDoS_TCP	10247	6.49%
9	Backdoor	10195	6.46%
10	Vulnerability_scanner	10076	6.39%
11	Port_Scanning	10071	6.38%
12	XSS	10052	6.37%
13	Password	9989	6.33%
14	MITM	1214	0.77%
15	Fingerprinting	1001	0.63%

Moreover, the Edge-IIoTset collected information from more than ten IoT sensors and actuators providing a diverse representation of real-world IoT environments; gathering a total of 157,800 network traffic. Covering 14 attack types along with normal traffic to create an imbalanced dataset as shown above in Table 3.

Table 4: Edge-IIoTset Attacks And Attacks Category

Attack Category	**Attack Type**
Distributed Denial-of-Service (DDoS)	TCP SYN flood DDoS
	UDP flood DDoS
	HTTP flood DDoS
	ICMP flood DDoS
Information Gathering	Port Scanning
	OS Fingerprinting

	Vulnerability Scanning
Man-in-the-middle (MITM)	DNS Spoofing
	ARP Spoofing
Injection	Cross-site Scripting (XSS)
	SQL Injection
	Uploading
Malware	Backdoor
	Password cracking
	Ransomware

Table 4 presents 15 types of attack covered in the paper of the Edge-IIoTset dataset [54], which are categorized into five attack groups. Notably, the reason for conflict between the 15 attacks mentioned in the paper and the 14 attacks observed in the dataset is because in the dataset the two attacks DNS Spoofing and ARP Spoofing were combined into a single MITM attack.

3.1.1. Attack Category 1: Distributed Denial-of-Service (DDoS)

Distributed Denial-of-Service (DDoS) is a type of cyber-attack where multiple compromised computers or devices, often referred to as "botnets," are used to flood a target system or network with a massive amount of traffic or requests, overwhelming the target and causing it to become unavailable or inaccessible to legitimate users [55]. In a DDoS attack, the attacking devices are distributed across various locations, making it difficult to identify and block the attack at its source. DDoS attacks typically aim to disrupt the normal operation of a website, server, or network by flooding it with traffic or requests beyond its capacity to handle. This can result in a temporary or prolonged outage, loss of revenue, damage to reputation, and potential financial or operational impacts for the targeted organization. DDoS attacks can be launched using various techniques, such as flooding the target with excessive network traffic, overwhelming its resources with a high volume of requests, or exploiting vulnerabilities in the target's systems or applications. DDoS attacks can also be combined with other types of attacks, such as malware infections or social engineering, to further disrupt or compromise the target's systems or data.

3.1.1.1. Attack 1: TCP SYN flood DDoS

TCP SYN flood is a type of DDoS attack that targets the TCP (Transmission Control Protocol) protocol, which is commonly used for establishing network connections between devices. In a TCP SYN flood attack, the attacker floods the target system with many synchronization (SYN) packets, which are the initial packets exchanged between devices to establish a TCP connection. In a typical TCP handshake process, the client sends a SYN packet to the server, the server responds with a synchronization-acknowledgment (SYN-ACK) packet, and the client completes the handshake by sending an ACK packet [56]. Once the handshake is completed, the TCP connection is established, and data can be exchanged between the client and server. However, in a TCP SYN flood attack, the attacker sends a high volume of SYN packets to the target system without completing the handshake process by sending the ACK packets. This floods the target system's resources, such as the system's TCP connection queue or memory, with half-open connections that are waiting for completion, consuming system resources and preventing legitimate clients from establishing connections. This can result in a denial of service, as the target system becomes overwhelmed and unable to respond to legitimate connection requests, causing disruption or unavailability of services. TCP SYN flood attacks are effective because they exploit the fundamental design of the TCP handshake process, which requires resources to be allocated for each half-open connection. These attacks can be challenging to mitigate, as they can be launched from multiple distributed sources, making it difficult to identify and block the attack traffic.

3.1.1.2. Attack 2: UDP flood DDoS

UDP flood is a type of DDoS attack that targets the UDP (User Datagram Protocol) protocol, which is a connectionless protocol used for transmitting data over a network. In a UDP flood attack, the attacker floods the target system with a high volume of UDP packets, overwhelming its resources and causing disruption or unavailability of services. Unlike TCP, which establishes a connection before transmitting data, UDP does not establish a connection and does not guarantee reliable delivery of packets [57]. This makes UDP flood attacks particularly effective, as they can be easily launched by sending a large number of UDP packets to the target system without the need for completing any handshake or connection setup

process. In a UDP flood attack, the attacker typically spoofs the source IP addresses of the packets, making it difficult to trace the attack back to its source. The target system's resources, such as its network bandwidth, processing power, or memory, can become overwhelmed as it tries to process the large volume of incoming UDP packets. This can result in a denial of service, as legitimate traffic may be unable to reach the target system, causing disruption or unavailability of services. UDP flood attacks can target specific UDP-based services or applications, such as DNS (Domain Name System) servers, VoIP (Voice over Internet Protocol) systems, online gaming servers, or other applications that rely on UDP for data transmission. These attacks can also be combined with other types of attacks, such as DNS amplification or reflection attacks, where the attacker exploits vulnerabilities in third-party systems to amplify the volume of attack traffic directed toward the target system.

3.1.1.3. Attack 3: HTTP flood DDoS

HTTP flood is a type of DDoS attack that targets web servers and web applications by overwhelming them with a high volume of HTTP (Hypertext Transfer Protocol) requests [58]. HTTP is the foundation of the World Wide Web and is used for transmitting data between web servers and clients, such as web browsers. In an HTTP flood attack, the attacker floods the target system with a massive number of HTTP requests, consuming its resources and causing disruption or unavailability of web services. HTTP flood attacks are often carried out using botnets, which are networks of compromised computers that are controlled remotely by the attacker. These botnets can generate a massive amount of HTTP requests from multiple sources, making it difficult to identify and block the attack traffic. The HTTP requests in an HTTP flood attack can be simple GET requests, which request a web page, or more complex POST requests, which send data to the web server. There are several variants of HTTP flood attacks, including volumetric HTTP flood attacks, which aim to overwhelm the target system's resources with a high volume of HTTP requests, and application-layer HTTP flood attacks, which aim to exploit vulnerabilities in web applications or web servers to cause disruption or unavailability of services. Application-layer HTTP flood attacks can target specific URLs, parameters, or functionalities of web applications, aiming to exhaust server-side resources, such as CPU, memory, or database connections, or trigger application-level errors or crashes.

3.1.1.4. Attack 4: ICMP flood DDoS

ICMP flood is a type of DDoS attack that targets the ICMP (Internet Control Message Protocol) protocol, which is a network protocol used for sending error messages and operational information about network conditions [59]. In an ICMP flood attack, the attacker floods the target system with a high volume of ICMP Echo Request (ping) packets, overwhelming its resources and causing disruption or unavailability of network services. ICMP flood attacks are often carried out using botnets, which are networks of compromised computers that are controlled remotely by the attacker. These botnets can generate a massive amount of ICMP Echo Request packets from multiple sources, making it difficult to identify and block the attack traffic. ICMP flood attacks can generate a large amount of network traffic and consume significant network bandwidth and processing power, leading to degradation or complete disruption of network services. ICMP flood attacks can be aimed at a specific target, such as a particular IP address or network, or they can be used as part of a larger DDoS attack targeting multiple systems or services. The attack traffic generated in an ICMP flood attack is typically characterized by a high volume of ICMP Echo Request packets, often with spoofed source IP addresses, making it challenging to trace the attack back to its source.

3.1.2. Attack Category 2: Information Gathering

Information-gathering attacks, also known as reconnaissance attacks, are a type of cyber-attack that involves gathering information about a target system or network with the intention of identifying vulnerabilities, weaknesses, or potential points of entry for further attacks [60]. Information-gathering attacks are often the first step in the cyber-attack chain and are carried out to gather intelligence and plan subsequent attacks. Information-gathering attacks can take various forms and can involve both passive and active techniques. Passive information-gathering techniques involve collecting data from publicly available sources, such as WHOIS databases, public websites, social media profiles, or online forums, without directly interacting with the target system or network. Active information-gathering techniques, on the other hand, involve actively probing or scanning the target system or network to collect information, such as network topology, open ports, running services, or system configurations. Information-gathering attacks can be conducted using automated tools, such as port scanners, vulnerability scanners, or reconnaissance frameworks, or manually by skilled

attackers who leverage their knowledge and expertise in identifying potential vulnerabilities or weaknesses in the target system or network. The information collected during the reconnaissance phase can be used to plan and launch subsequent attacks, such as exploitation of vulnerabilities, password attacks, or social engineering attacks, with the ultimate goal of gaining unauthorized access, stealing sensitive data, or disrupting the target system or network.

3.1.2.1. Attack 5: Port Scanning

Port scanning is a type of cyber-attack that involves probing a target system or network to identify open ports, which are network communication endpoints, and determine the services or applications that are listening on those ports [61]. Port scanning is commonly used as a reconnaissance technique by attackers to gather information about a target system's network topology, identify potential vulnerabilities, and plan further attacks. Port scanning attacks can be carried out using automated tools or manually by skilled attackers. Automated port scanning tools, such as Nmap, are widely available and can scan a range of IP addresses or a single host for open ports in a short amount of time. These tools can scan for common TCP or UDP ports, such as FTP (File Transfer Protocol), SSH (Secure Shell), HTTP, or DNS (Domain Name System), among others. Port scanning attacks can help attackers identify potential vulnerabilities in the target system or network, which can then be exploited in subsequent attacks to gain unauthorized access, steal sensitive data, or disrupt services.

3.1.2.2. Attack 6: OS Fingerprinting

OS (Operating System) fingerprinting is a technique used in cybersecurity to determine the type or version of the operating system running on a target system or network device. This information can be useful for attackers to identify potential vulnerabilities or weaknesses in the target system and tailor subsequent attacks accordingly [62]. OS fingerprinting can be classified into various types based on the techniques used. One type of OS fingerprinting is active fingerprinting, where the attacker sends probes or packets to the target system or network device and actively analyzes the responses to determine the operating system. This may involve sending specific requests or queries to services or applications running on the target system and analyzing the responses or analyzing network traffic patterns. Another type

of OS fingerprinting is passive fingerprinting, where the attacker captures and analyzes network traffic between the target system and other devices or systems to determine the operating system based on patterns or characteristics observed in the traffic. Passive fingerprinting does not involve actively sending probes or packets to the target system. There is also inference-based fingerprinting, where the attacker uses statistical or machine learning techniques to analyze patterns or characteristics in the network traffic or responses from the target system to infer the type or version of the operating system. OS fingerprinting attacks can provide valuable information to attackers about the target system's operating system, which can be used to launch subsequent attacks that are tailored to exploit known vulnerabilities or weaknesses in that particular operating system.

3.1.2.3. Attack 7: Vulnerability Scanning

Vulnerability scanning is a type of cyber-attack that involves systematically scanning a target system or network for known vulnerabilities or weaknesses that could be exploited by attackers. This type of attack typically involves using automated tools or software to scan for vulnerabilities in networks, systems, applications, or services. Vulnerability scanning attacks can be performed in different ways. One approach is network vulnerability scanning, where the attacker scans the target network for known vulnerabilities in network devices, such as routers, switches, and firewalls, or in network protocols, such as TCP/IP or DNS, that could be exploited to gain unauthorized access or disrupt network operations. Another approach is system vulnerability scanning, where the attacker scans the target system, such as servers, workstations, or mobile devices, for known vulnerabilities in the operating system, software applications, or services running on the system, that could be exploited to gain unauthorized access or compromise the integrity or confidentiality of data. Lastly, application vulnerability scanning involves scanning the target applications, such as web applications or databases, for known vulnerabilities in the code, configuration, or input validation that could be exploited to gain unauthorized access, execute arbitrary code, or steal sensitive information [63]. Vulnerability scanning attacks can help attackers identify potential weaknesses in the target system or network that can be exploited to gain unauthorized access, steal information, disrupt operations, or launch further attacks.

3.1.3. Attack Category 3: Man-in-the-middle (MITM)

Man-in-the-middle (MITM) is a type of cyber-attack where an attacker intercepts communication between two parties, such as a client and a server, in order to eavesdrop, modify, or inject malicious content into the communication without the knowledge or consent of the parties involved. The attacker positions themselves between the legitimate parties and can intercept, modify, or redirect the communication in real-time, giving them the ability to capture sensitive information, manipulate data, or impersonate one of the parties [64]. MITM attacks can occur in various ways, such as passive or active MITM. In a passive MITM attack, the attacker simply eavesdrops on the communication between the two parties without modifying the content, intercepting, and capturing data transmitted over a network, such as passwords or credit card numbers. In an active MITM attack, the attacker not only intercepts the communication but also modifies the content or injects malicious content into the communication, such as altering data packets, injecting malware, or malicious scripts, or redirecting communication to malicious servers or websites. MITM attacks can be executed in different contexts, such as in wired or wireless networks, on public Wi-Fi networks, or on unsecured or compromised network devices, such as routers or switches. These attacks can be used to gain unauthorized access to sensitive information, steal credentials, perform fraudulent activities, or launch further attacks, such as session hijacking, data manipulation, or eavesdropping on confidential communication.

3.1.3.1. Attack 8: DNS Spoofing

DNS spoofing is a type of cyber-attack where an attacker falsifies the Domain Name System (DNS) resolution process to redirect users to malicious websites or intercept their communication. DNS is responsible for translating human-readable domain names, such as www.example.com, into IP addresses, which are the numerical addresses that computers use to identify each other on the internet [65]. In a DNS spoofing attack, the attacker manipulates the DNS resolution process by providing false or misleading information to DNS servers or clients. This can involve creating fake DNS responses that contain incorrect IP addresses, domain names, or other DNS data, or intercepting legitimate DNS responses and modifying them in transit. By doing so, the attacker can redirect users to malicious websites that may look legitimate but are designed to steal sensitive information, inject malware, or perform

other malicious activities. DNS spoofing attacks can be carried out in various ways, such as by poisoning the DNS cache of a DNS server, redirecting DNS queries to malicious DNS servers, or manipulating the DNS responses at the network level using techniques like ARP spoofing or DNS packet injection. DNS spoofing can have serious consequences, as it can enable attackers to intercept sensitive information, perform phishing attacks, launch further attacks, such as man-in-the-middle attacks, or gain unauthorized access to systems or networks.

3.1.3.2. Attack 9: ARP Spoofing

ARP (Address Resolution Protocol) spoofing, also known as ARP poisoning or ARP cache poisoning, is a type of cyber-attack where an attacker falsifies the MAC (Media Access Control) address associations in the ARP cache of a network device to intercept, modify, or redirect network traffic [66]. ARP is a protocol used in local area networks (LANs) to map IP addresses to MAC addresses, which are unique hardware addresses associated with network devices, such as network interface cards (NICs). When devices need to communicate on a LAN, they use ARP to request and store the MAC address of the intended destination device in their ARP cache for future reference, avoiding the need to perform repetitive IP-to-MAC address resolutions. In an ARP spoofing attack, the attacker sends falsified ARP messages, either gratuitous or in response to ARP requests, with forged MAC address associations to the victim device or other devices on the same LAN. This causes the victim device or other devices to update their ARP caches with incorrect MAC address information, associating the attacker's MAC address with the IP addresses of legitimate devices, such as the default gateway or other hosts. As a result, network traffic intended for these legitimate devices is redirected to the attacker's device, allowing the attacker to intercept, modify, or eavesdrop on the traffic. ARP spoofing attacks can be used to launch various types of attacks, such as man-in-the-middle attacks, where the attacker can intercept and modify network traffic between two legitimate devices, perform session hijacking, inject malicious content, or capture sensitive information. ARP spoofing attacks can also be used to conduct reconnaissance, gain unauthorized access to systems or networks, or disrupt network communication.

3.1.4. Attack Category 4: Injection

An injection attack is a type of cyber-attack where an attacker inserts malicious code or data into an application or system with the intention of executing unauthorized commands, manipulating data, or gaining unauthorized access to the system [67]. Injection attacks typically target vulnerable points in an application's input validation or parameter handling mechanisms to exploit weaknesses and bypass security measures. Injection attacks can occur in various forms, such as SQL injection, NoSQL injection, LDAP injection, XML injection, command injection, and JavaScript injection, among others. These attacks exploit vulnerabilities in applications that allow untrusted data, such as user input or other external data, to be included in a query or command without proper validation or sanitization. As a result, the malicious code or data is interpreted as legitimate and executed by the application or system, allowing the attacker to gain unauthorized access, extract sensitive information, modify data, or execute arbitrary commands. Injection attacks can have serious consequences, as they can result in data breaches, system compromise, unauthorized access, data manipulation, and other malicious activities.

3.1.4.1. Attack 10: Cross-site Scripting (XSS)

Cross-site Scripting (XSS) is a type of web application vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. XSS occurs when an application fails to properly validate and sanitize user input, allowing malicious scripts to be executed within the context of a legitimate website or web application [68]. There are two main types of XSS attacks: reflected XSS and stored XSS. In reflected XSS, the malicious script is included in a URL or in the parameters of a web form and then reflected back to the user in the HTML response. When the user's browser renders the HTML response, the malicious script is executed, allowing the attacker to steal sensitive information, manipulate content, or perform other malicious actions on the user's behalf. In stored XSS, the malicious script is stored in a database or other storage location within the application, and then retrieved and displayed to other users when they access the affected page. This allows the attacker to inject malicious scripts that persistently affect multiple users, potentially leading to widespread damage and data theft. XSS attacks can have serious consequences, including data theft,

unauthorized access, cookie stealing, session hijacking, defacement of web pages, and other malicious activities.

3.1.4.2. Attack 11: SQL Injection

SQL Injection is a type of web application vulnerability that allows an attacker to manipulate a website's SQL database by injecting malicious SQL queries through user input fields or other vulnerable areas of the application. SQL Injection occurs when an application fails to properly validate and sanitize user input before using it to construct SQL queries, allowing the attacker to modify or execute unintended SQL queries on the database [69]. There are several types of SQL Injection attacks that attackers can exploit to manipulate and extract data from vulnerable web applications. In-band SQL Injection is a type of attack where the attacker injects malicious SQL queries directly into the user input field or other vulnerable areas of the application, and the results are returned in the application's response. This type of attack is also known as "error-based" or "union-based" SQL Injection. Blind SQL Injection, on the other hand, is a type of attack where the attacker does not receive direct feedback from the application about the results of the injected SQL queries. Instead, the attacker uses techniques such as time delays or Boolean-based queries to infer the results indirectly, allowing them to extract data from the database. Out-of-band SQL Injection is another type of attack where the attacker does not receive the results of the injected SQL queries through the application's response, but rather through a separate communication channel, such as email or a separate web service. This type of attack is also known as "out-of-band" or "second-order" SQL Injection. SQL Injection attacks can have serious consequences, including unauthorized access to sensitive data, modification or deletion of data, privilege escalation, and other malicious activities.

3.1.4.3. Attack 12: Uploading

In an uploading attack, the attacker may take advantage of vulnerabilities in a web application or other file upload functionality to bypass security measures and upload malicious files. Once the malicious files are uploaded, the attacker may be able to execute them, potentially gaining unauthorized access, compromising data, or causing other types of harm [70]. The specifics of an uploading attack may vary depending on the system or

application being targeted, the techniques used by the attacker, and the objectives of the attack. Common methods to defend against uploading attacks include implementing proper input validation and file upload validation, ensuring secure configuration of file upload functionality, restricting access permissions for uploaded files, and regularly monitoring and auditing system activity for signs of suspicious behaviour. It's important to note that cybersecurity attacks are constantly evolving, and new attack techniques and vulnerabilities may arise over time.

3.1.5. Attack Category 5: Malware

Malware, short for malicious software, refers to any software specifically designed to harm, exploit, or compromise the security of computer systems, networks, or devices [71]. Malware is a broad category that includes various types of malicious software, such as viruses, worms, trojan horses, ransomware, adware, spyware, and other malicious programs. Malware can be distributed through various means, including infected email attachments, malicious websites, compromised software, social engineering attacks, and other methods. Once installed on a victim's system or device, malware can carry out a wide range of malicious activities, such as stealing sensitive information, disrupting system operations, modifying or deleting data, hijacking system resources, conducting unauthorized activities, or providing unauthorized access to remote attackers. Malware is a serious cybersecurity threat that can cause significant damage to individuals, organizations, and even entire networks.

3.1.5.1. Attack 13: Backdoor

A backdoor attack refers to a malicious activity where an unauthorized entry point or hidden access point is created in a system or network, allowing an attacker to gain unauthorized access and control over the system or network. A backdoor is typically created by exploiting vulnerabilities or weaknesses in the system or network, and it provides a secret and unauthorized entry point for the attacker to bypass normal authentication and gain unauthorized access to the system or network [72]. Backdoor attacks can be carried out through various means, including software vulnerabilities, social engineering, malware, or other malicious techniques. Once a backdoor is successfully implanted, the attacker can use it to gain unauthorized access, execute commands, manipulate data, steal information, disrupt

operations, or carry out other malicious activities without being detected. Backdoor attacks are considered highly dangerous as they provide unauthorized access to attackers, allowing them to maintain control over the compromised system or network for an extended period of time. Backdoors can be difficult to detect, as they are typically designed to blend in with legitimate system components or activities, making them challenging to identify through traditional security measures.

3.1.5.2. Attack 14: Password cracking

Password cracking attack, also known as password hacking, is a type of cyber-attack in which an attacker attempts to gain unauthorized access to a system, network, or account by guessing or systematically cracking the passwords used for authentication [73]. Passwords are commonly used as a form of authentication to protect access to various resources, such as user accounts, databases, applications, and systems. There are several methods that attackers may use to carry out password cracking attacks. One method is brute force attack, in which the attacker systematically tries every possible combination of characters until the correct password is guessed. This method can be time-consuming and resource-intensive, but it can be effective if the password is weak or short. Another method is dictionary attack, in which the attacker uses a list of known words or commonly used passwords, known as a "dictionary," to systematically try each word in the list as a potential password. This method is more efficient than brute force as it targets commonly used passwords, but it may not be effective against complex or unique passwords. Rainbow table attack is another method, in which the attacker uses precomputed tables, known as "rainbow tables," that contain hashes of commonly used passwords and their corresponding plaintext values. The attacker compares the hashes of the target passwords with the hashes in the rainbow tables to quickly identify matches and obtain the plaintext passwords. Hybrid attack is a method in which the attacker combines various techniques, such as brute force, dictionary, and rainbow table attacks, to increase the chances of success in cracking passwords. Social engineering attack is also a method, in which the attacker manipulates or tricks individuals into revealing their passwords through techniques such as phishing, pretexting, or other forms of social engineering. Password cracking attacks can be highly effective if passwords are weak, easily guessable, or improperly stored.

3.1.5.3. Attack 15: Ransomware

Ransomware is a type of malicious software (malware) that encrypts or otherwise restricts access to a victim's files or computer system and demands a ransom in exchange for restoring access [74]. Ransomware attacks typically involve the use of encryption algorithms to lock files, making them inaccessible to the victim without the decryption key held by the attacker. Once the files are encrypted, the attacker typically displays a ransom message on the victim's screen, providing instructions on how to pay the ransom and obtain the decryption key. Ransomware attacks can have severe consequences, as they can cause data loss, disrupt business operations, and result in financial and reputational damage. Ransomware can be delivered through various methods, including phishing emails, malicious attachments or links, drive-by downloads from compromised websites, or via infected USB drives or other removable media. There are different types of ransomware, including encrypting ransomware that encrypts files, and locker ransomware which restricts access to the victim's system without encrypting files. Some ransomware variants also use other techniques such as data theft, where the attacker exfiltrates sensitive data from the victim's system before encrypting files and threatens to release the data if the ransom is not paid.

3.2. Background Information About Techniques Used

In the development of a machine learning algorithm, selecting the appropriate methods and techniques is crucial as it directly impacts the model's performance [75]. By reviewing the existing literature related to the Edge-IIoTset dataset [54], it was observed that none of the prior studies employed an optimization algorithm for optimizing the attributes of machine learning algorithms. This notable gap in the literature underscores the potential of employing an optimization algorithm to solve the issue of finding a good attribute of machine learning algorithm components.

In light of this gap in the literature and the necessity for effective optimization methods, the utilization of neural networks stands out as a promising approach, considering their ability to learn complex patterns from data [76]. However, the performance of neural networks heavily relies on selecting appropriate hyperparameters and optimizing model

weights. Traditional optimization techniques, such as grid search or random search, are often computationally expensive and may not yield optimal solutions in high-dimensional spaces.

To address this challenge, the application of genetic algorithms presents an intriguing solution. Genetic algorithms leverage principles inspired by natural selection to efficiently search through the solution space and identify optimal or near-optimal solutions [77]. Employing genetic algorithms to optimize the weights of neural networks, can leverage the ability to efficiently explore the solution space and find solutions that maximize the performance of the model.

Therefore, this study proposes the utilization of neural networks as the underlying machine learning algorithm for the Edge-IIoTset dataset, coupled with the optimization of model weights using genetic algorithms. This approach not only addresses the gap in the literature but also offers a novel and effective methodology for enhancing the performance of machine learning models in industrial IoT environments.

3.2.1. Neural Network

A neural network is a computational model inspired by the structure and functionality of the human brain. It's a fundamental algorithm in machine learning, particularly in the domain of deep learning [76], [78]. The basic architecture consists of interconnected nodes, organized into layers, including an input layer, one or more hidden layers, and an output layer as shown below in Figure 1.

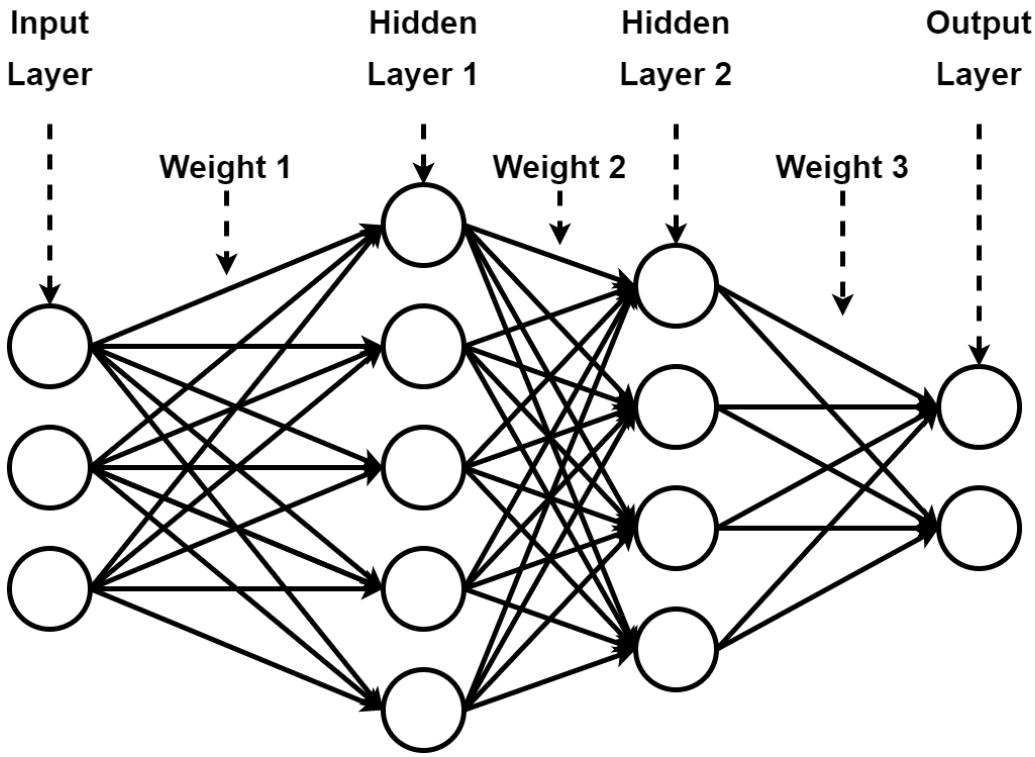


Figure 1: Neural Network Example With Two Hidden Layers

Each connection between nodes is associated with a weight, and the network learns by adjusting these weights during the training process. The neural network employs activation functions to introduce non-linearities, allowing it to model complex relationships and capture intricate patterns within the data. Training a neural network involves optimizing these weights through a process known as backpropagation [79]. During backpropagation, the network compares its predictions to the actual outcomes, calculates the error, and then adjusts the weights to minimize this error. In the context of deep learning, neural networks with multiple hidden layers are often referred to as deep neural networks. These deep architectures have demonstrated remarkable success in various applications, including image and speech recognition, natural language processing, and even playing strategic games [78], [80]. One of the strengths of neural networks lies in their ability to automatically learn hierarchical representations of data, enabling them to discern intricate features and relationships. The adaptability and capacity to learn from vast amounts of data make neural networks a powerful tool for solving complex problems across diverse domains in modern machine learning.

Neural networks are increasingly employed in the detection of cyber-attacks due to their ability to learn complex patterns and anomalies within network data [81]. The neural network algorithm proves highly beneficial for the detection of cyber-attacks due to its inherent capacity to learn complex patterns, adapt to evolving threats, and provide a sophisticated framework for analyzing vast and dynamic datasets. By leveraging deep learning architectures, neural networks excel in discerning subtle anomalies and patterns within network traffic, enabling the identification of potential cyber-threats with remarkable accuracy. Their ability to automatically extract relevant features and hierarchically represent intricate relationships within data contributes to a robust cyber defence mechanism. Furthermore, neural networks contribute to real-time threat detection, reducing false positives, and enhancing the overall cybersecurity posture by providing a proactive and adaptive approach to combating cyber-threats in today's rapidly evolving digital landscape. The below points illustrate the effectiveness of the neural network algorithm in detecting cyber-attacks [82]:

1. **Pattern Recognition:** Neural networks excel at recognizing patterns in data, making them well-suited for identifying unusual or malicious activities within network traffic. By training on normal behaviour, the network can later detect deviations that may indicate a cyber-attack.
2. **Anomaly Detection:** Neural networks can be trained in an unsupervised manner to recognize anomalies. In the context of cybersecurity, deviations from normal network behaviour can be flagged as potential cyber threats. Unusual patterns, unexpected data flows, or suspicious activities can trigger alerts.
3. **Feature Extraction:** Neural networks automatically learn to extract relevant features from raw network data. This is particularly valuable in cybersecurity, where identifying distinctive features of various types of attacks is essential. The network can learn to recognize patterns indicative of specific attack types.
4. **Adaptability to Dynamic Threats:** Cyber threats are constantly evolving, requiring adaptive detection mechanisms. Neural networks, especially deep learning models, can adapt to new attack patterns without explicit reprogramming. This flexibility is crucial in the ever-changing landscape of cybersecurity.
5. **Multi-Layered defence:** Just as neural networks have multiple hidden layers for hierarchical feature extraction, they contribute to a multi-layered defense strategy in

cybersecurity. Each layer of the network can focus on detecting specific aspects of cyber threats, enhancing overall detection capabilities.

6. **Handling Complex Data:** Neural networks can effectively handle the complexity of various types of data generated in network traffic, such as packet headers, payloads, and temporal patterns. This makes them versatile in detecting sophisticated attacks that might involve multiple stages or techniques.
7. **False Positive Reduction:** By learning from a diverse set of normal network behaviors, neural networks can help reduce false positives in cyber-attack detection. Their ability to discern between normal and abnormal patterns contributes to more accurate and efficient threat identification.

3.2.2. Genetic Algorithm

The Genetic Algorithm (GA) operates on a foundational framework that mimics the principles of natural selection and genetics, offering a robust approach to solving intricate optimization and search challenges [83]. This heuristic optimization algorithm is inspired by the enduring process of natural evolution, where adaptation and selection lead to the emergence of optimal traits over successive generations. Genetic Algorithms stand out for their ability to efficiently explore vast solution spaces, tackle complex optimization challenges, and adaptively evolve solutions over successive generations. This versatility has positioned GAs as a valuable tool in various domains, ranging from machine learning and optimization problems to combinatorial optimization tasks, showcasing their effectiveness in addressing diverse real-world problems.

Components of Genetic Algorithms [84]**:**

- **Initialization:** The algorithm commences by establishing a diverse population of potential solutions. This population is often initialized randomly or through predefined methods, representing a varied set of potential outcomes.
- **Fitness Evaluation:** Each individual in the population undergoes scrutiny through an objective function. This function quantifies the fitness of a solution by measuring how well it aligns with the predefined optimization criteria. Higher fitness indicates a greater likelihood of survival.

- **Selection:** Imitating the evolutionary principle of "survival of the fittest," individuals are chosen for reproduction based on their fitness levels. Higher-fitness individuals stand a better chance of contributing their genetic material to the next generation.
- **Crossover (Recombination):** Pairs of selected individuals engage in crossover, a process mirroring genetic recombination in biological reproduction. This exchange of genetic information generates offspring endowed with a blend of traits inherited from both parents.
- **Mutation:** Introducing an element of randomness, a genetic mutation occurs, bringing about random changes to the genetic makeup of select individuals. This injects diversity into the population, preventing stagnation and encouraging the exploration of new solution spaces.
- **Replacement:** The existing generation is replaced by the newly generated one, comprising offspring resulting from crossover and mutated individuals. This fresh cohort is subjected to fitness evaluation, perpetuating the evolutionary cycle.
- **Termination Criteria:** The algorithm continually checks for termination criteria, which could include a predefined number of generations, attainment of a satisfactory solution, or the achievement of a convergence threshold. Termination criteria ensure that the algorithm concludes when specific conditions are met.
- **Output:** The culmination of the algorithmic process is the extraction of the solution that optimally satisfies the defined criteria or fulfills the termination conditions, providing a refined and effective outcome.

Genetic algorithms play a crucial role in enhancing the efficacy of cyber-attack detection systems by mimicking the principles of natural selection and evolution within the realm of computer security. In the context of cybersecurity, a genetic algorithm operates by generating a diverse set of potential solutions, represented as strings of binary code, to address the evolving nature of cyber threats. These solutions, analogous to individual organisms in nature, undergo a process of selection, crossover, and mutation to produce successive generations of candidate solutions. By evaluating the fitness of each solution based on its ability to detect and respond to specific cyber-attack patterns, the genetic algorithm refines and evolves the population over iterations, favouring traits that demonstrate better performance against

emerging threats. This adaptive and iterative process allows the genetic algorithm to continuously optimize the detection mechanisms, enabling cybersecurity systems to adapt dynamically to the ever-changing landscape of cyber threats.

3.2.3. Principal Component Analysis

Principal Component Analysis (PCA) is a widely employed technique used in the field of machine learning and data analysis, known for its ability to reduce the dimensionality of large datasets while preserving as much of the original variance as possible [85]. PCA is particularly useful when dealing with high-dimensional data, where the presence of many correlated features can lead to challenges such as overfitting and computational inefficiency.

The core idea behind PCA is to transform the original set of correlated variables into a new set of uncorrelated variables called principal components [86]. These principal components are linear combinations of the original variables and are ordered by the amount of variance they explain in the data. The first principal component captures the maximum variance, while each subsequent component accounts for the remaining variance under the constraint that it is orthogonal to the preceding components.

The process of PCA involves the following steps [87]:

1. **Standardization**: The data is first standardized, ensuring that each feature contributes equally to the analysis by subtracting the mean and scaling to unit variance.
2. **Covariance Matrix Computation**: The covariance matrix of the standardized data is computed, capturing the relationships between different features.
3. **Eigenvalue Decomposition**: The eigenvalues and eigenvectors of the covariance matrix are calculated. The eigenvectors represent the directions of the principal components, while the eigenvalues indicate the magnitude of variance explained by each component.
4. **Selection of Principal Components**: The principal components are ranked according to their eigenvalues, and a subset of the top components is selected, depending on the desired level of dimensionality reduction.

5. **Projection**: The original data is projected onto the selected principal components, resulting in a transformed dataset with reduced dimensionality.

PCA offers several advantages in machine learning applications:

- **Dimensionality Reduction**: By reducing the number of features, PCA helps mitigate the risk of overfitting and enhances the efficiency of machine learning algorithms.
- **Noise Reduction**: PCA can filter out noise by discarding components associated with low variance, thereby improving the model's performance.
- **Feature Interdependence**: PCA eliminates multicollinearity by generating uncorrelated principal components, making it easier to interpret the relationships within the data.

3.3. Implementation

The following section will provide a detailed illustration of how the proposed hybrid algorithm functions. It will encompass the entire process from data loading to model training and evaluation.

Initially, the section describes the extraction and preprocessing of data from the dataset, which includes encoding categorical features and labels and selecting a stratified sample to maintain class balance. Then the implementation of a dimension reduction technique using PCA to retain 95% of the variance. This is followed by partitioning the data into training and testing sets. Subsequently, the creation and configuration of the hybrid algorithm are discussed, including the definition of a neural network structure and the application of a genetic algorithm for parameter optimization.

3.3.1. Data Preparation and Sampling

Dealing with industrial devices presents challenges, primarily due to the enormous amount of data they generate [88]. Therefore, using a sampling technique is a crucial step to enhance the machine learning algorithm efficiency without impacting the accuracy [89]. After looking into various sampling techniques, the stratified sampling technique was chosen for the Edge-IIoTset dataset due to its inherent class imbalance. This method ensures that each attack type is adequately represented in both training and testing sets, preventing bias and

promoting model generalization [90]. By preserving the proportional representation of attack types, it facilitates balanced model evaluation across different categories as shown below in Figure 2, enhancing model efficacy and reliability.

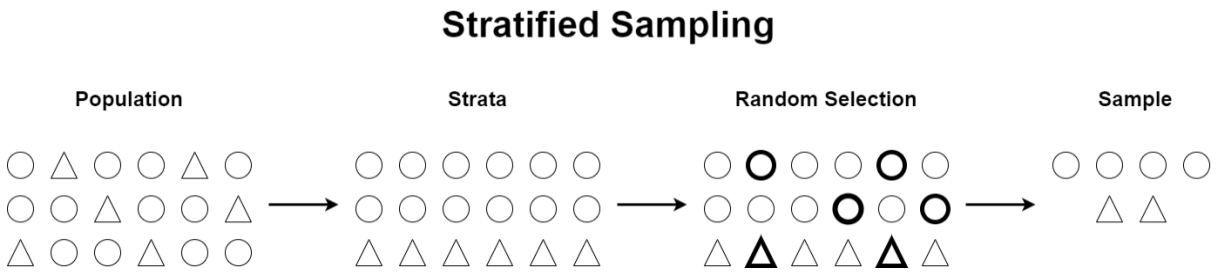


Figure 2: Stratified Sampling

Starting the preprocessing phase by loading the ML-EdgeIIoT-dataset.csv using the pd.read_csv function provided by the Pandas library with a “low_memory = False” parameter to instruct the system to load the entire dataset into memory at once, ensuring efficient processing.

After loading the data, the initial step involved extracting the features, focusing on the first 61 columns. Then, these features were converted from categorical variables to numerical representations by assigning a unique numerical identifier to each distinct value within the features, enabling machine learning algorithms to process them effectively. Similarly, the attack categories went for a similar transformation, employing a map function to assign numerical values for each attack type for memorization and efficiency.

Lastly, a stratified sample is generated using the train_test_split function from the sklearn.model_selection module. This function partitions the dataset into training and testing subsets while preserving the distribution of the target variable “Attack_type”. The stratification ensures that each subset maintains the same proportion of different attack types as the original dataset, thereby preventing bias in the model evaluation process. The size of the testing subset has been set to 20% of the original dataset, and the algorithm was trained three times using different randomly chosen random states (7, 10, and 42) to ensure varied sampling for consistency in the results.

Table 5: Dataset Sampling Using The Stratified Method

N	Traffic Type	Original Amount	Sample Amount	Percentage
1	Normal	24301	4860	15.40%
2	DDoS_UDP	14498	2899	9.19%
3	DDoS_ICMP	14090	2818	8.93%
4	Ransomware	10925	2185	6.92%
5	DDoS_HTTP	10561	2112	6.69%
6	SQL_injection	10311	2062	6.53%
7	Uploading	10269	2053	6.51%
8	DDoS_TCP	10247	2049	6.49%
9	Backdoor	10195	2039	6.46%
10	Vulnerability_scanner	10076	2015	6.39%
11	Port_Scanning	10071	2014	6.38%
12	XSS	10052	2010	6.37%
13	Password	9989	1997	6.33%
14	MITM	1214	242	0.77%
15	Fingerprinting	1001	200	0.63%

Table 5 presents the distribution of traffic types in both the original dataset and the generated stratified sample. This table illustrates how the stratification process preserves the proportions of different attack types in the sample, ensuring a representative subset for model training and evaluation. By maintaining consistency in the distribution of attack types, the stratified sample mitigates the risk of bias during model assessment.

3.3.2. Feature Selection

The ML-EdgeIIoT dataset has 63 columns as shown below in Table 7, where the initial 61 columns represent various features extracted from the IIoT traffic data, and the last two represent the attack label and attack type. However, the extensive number of features poses challenges in terms of redundancy and importance. Given this scenario, employing a feature reduction technique becomes imperative to streamline the dataset and enhance model efficiency.

Features:

Table 6: EdgeIIoT Columns

N	Columns
1	frame.time
2	ip.src_host
3	ip.dst_host
4	arp.dst.proto_ipv4
5	arp.opcode
6	arp.hw.size
7	arp.src.proto_ipv4
8	icmp.checksum
9	icmp.seq_le
10	icmp.transmit_timestamp
11	icmp.unused
12	http.file_data
13	http.content_length
14	http.request.uri.query
15	http.request.method
16	http.referer
17	http.request.full_uri
18	http.request.version
19	http.response
20	http.tls_port
21	tcp.ack
22	tcp.ack_raw
23	tcp.checksum
24	tcp.connection.fin
25	tcp.connection.rst
26	tcp.connection.syn
27	tcp.connection.synack
28	tcp.dstport

29	tcp.flags
30	tcp.flags.ack
31	tcp.len
32	tcp.options
33	tcp.payload
34	tcp.seq
35	tcp.srcport
36	udp.port
37	udp.stream
38	udp.time_delta
39	dns.qry.name
40	dns.qry.name.len
41	dns.qry.qu
42	dns.qry.type
43	dns.retransmission
44	dns.retransmit_request
45	dns.retransmit_request_in
46	mqtt.conack.flags
47	mqtt.conflag.cleansess
48	mqtt.conflags
49	mqtt.hdrflags
50	mqtt.len
51	mqtt.msg_decoded_as
52	mqtt.msg
53	mqtt.msgtype
54	mqtt.proto_len
55	mqtt.protoname
56	mqtt.topic
57	mqtt.topic_len
58	mqtt.ver
59	mbtcp.len

60	mbtcp.trans_id
61	mbtcp.unit_id
62	Attack_label
63	Attack_type

After conducting extensive research on various dimension reduction techniques, it became evident that selecting the most suitable method was crucial for optimizing the ML-EdgeIIoT dataset. Upon thorough examination and comparison, PCA emerged as the most fitting choice for several reasons [87], [91], [92].

Firstly, PCA is adept at identifying and capturing the underlying patterns and correlations within high-dimensional data by transforming it into a lower-dimensional space. This reduction in dimensionality not only simplifies the dataset but also retains the essential information embedded within the features [93]. Moreover, PCA facilitates the extraction of orthogonal components, known as principal components, which represent the directions of maximum variance in the original feature space. By prioritizing these principal components, PCA effectively highlights the most significant sources of variation in the data while minimizing information loss.

Additionally, PCA offers interpretability by providing insights into the relative importance of each feature in contributing to the variance within the dataset. This attribute enables researchers to discern the key drivers influencing the IIoT traffic patterns, thereby facilitating better decision-making in feature selection and model development. Furthermore, PCA's ability to mitigate multicollinearity among features is particularly advantageous in the ML-EdgeIIoT dataset, where certain features may exhibit high intercorrelation. By reducing multicollinearity, PCA enhances the stability and robustness of subsequent machine-learning models, thereby improving their predictive performance [94].

As shown in the below Figure, the process begins by computing the mean of each feature in the dataset and then centering the data by subtracting the mean from each data point. Next, PCA calculates the covariance matrix, which describes the relationships between

different features and helps identify patterns in the data. After that, the eigenvalue decomposition is performed on the covariance matrix to find its eigenvalues and eigenvectors, representing the principal components. These components are sorted based on the importance of capturing variance in the data. PCA computes the explained variance ratio for each principal component, indicating the proportion of variance explained by each component relative to the total variance. Finally, PCA projects the original data onto the selected principal components, transforming it into a lower-dimensional space while preserving as much information as possible.

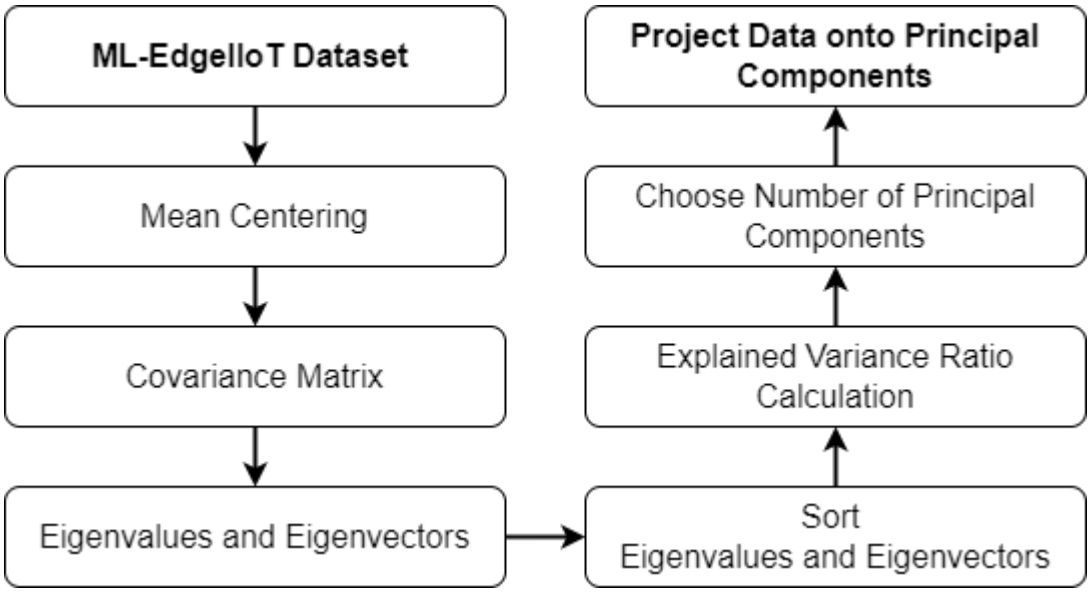


Figure 3: PCA Steps

To determine the optimal number of Principal Components (PCs) while retaining significant information, a threshold of 95% was selected, aligning with established findings from various research studies [95], [96], [97]. This choice is supported by the "elbow method," as depicted in the accompanying figure, which demonstrates the point where the explained variance begins to plateau. This empirical evidence reinforces the selection of 95% as the threshold for preserving substantial data integrity.

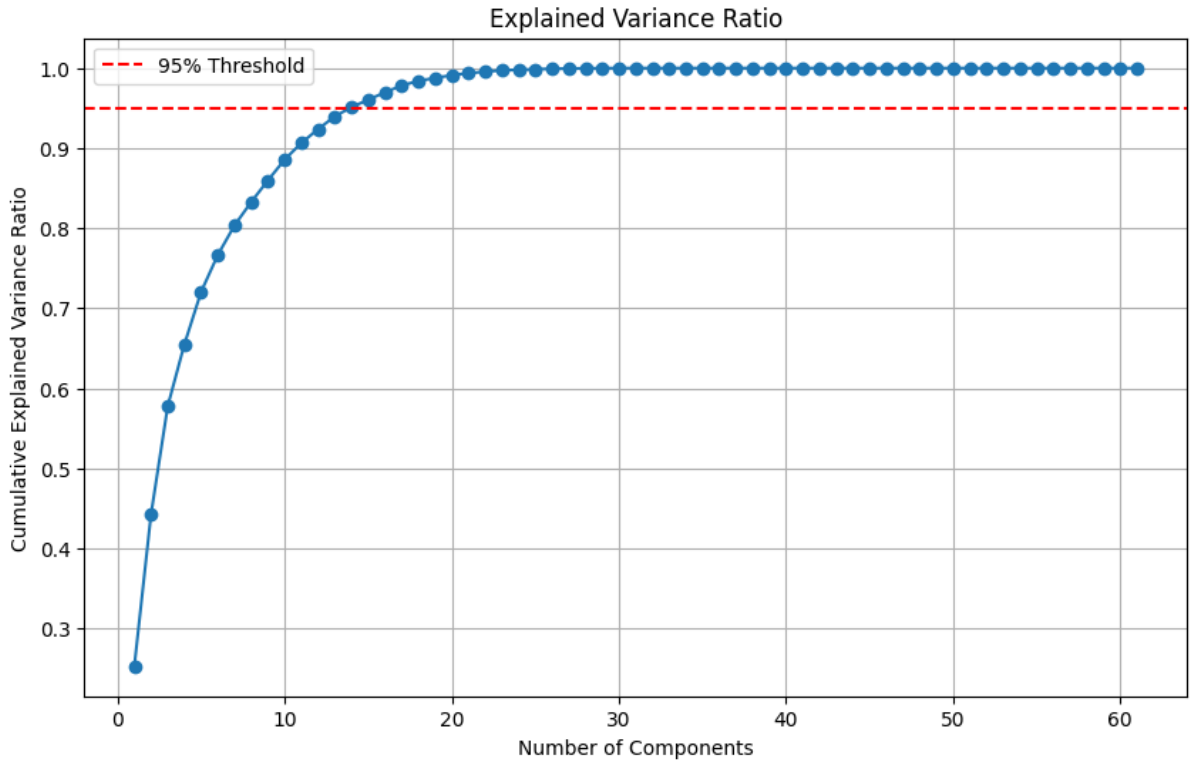


Figure 4: PCA Threshold Selection

By performing PCA on the dataset, its dimensionality has been reduced while preserving most of its variance, as shown in the table below, which displays the explained variance ratio for each extracted principal component. It shows that the first principal component accounts for 25.29% of the total variance in the data, followed by the second component with 18.99%. In total, 95% of the dataset's variance can be explained by the first 14 principal components, as indicated by the cumulative explained variance ratio. This analysis helps in understanding the relative importance of each principal component in representing the original dataset's variability.

Table 7: Selected PCA

Principal Component	Explained Variance Ratio
1	0.2529
2	0.1899
3	0.1351
4	0.0774
5	0.0655

6	0.0461
7	0.0369
8	0.0292
9	0.0269
10	0.026
11	0.0206
12	0.0173
13	0.0159
14	0.0115

The tables in Appendix A present the loadings of original features onto the principal components obtained through PCA. Each row corresponds to an original feature, while each column represents a principal component. The values in the table indicate the contribution of each original feature to each principal component. Positive values signify a positive correlation between the original feature and the principal component, while negative values indicate a negative correlation. The table allows for the interpretation of the structure and composition of each principal component, aiding in understanding the underlying patterns and relationships within the data. This comprehensive analysis facilitates dimensionality reduction and feature extraction while preserving the most significant information contained in the dataset.

3.3.3. Hybrid Algorithm

The hybrid algorithm of neural network and genetic algorithm starts by creating a class called NeuralNetworkGA. The class is composed of thirteen main functions (Initialization, Parameter Initialization, Activation Functions (sigmoid and softmax), Forward Propagation, Prediction, Population Initialization, Fitness Calculation, Genome Reshaping, Crossover, Mutation, and Optimization Loop, and Training.

1. **Initialization:**

   In the first function of the code, the initialization function serves as the cornerstone for initializing the hybrid algorithm of neural network and genetic algorithm

parameters. These parameters include the input size, hidden layer sizes, output size, population size, elite size, number of generations, mutation rate, learning rate, and parameters.

2. **Parameter Initialization:**

   The parameter initialization process involves the creation of initial weight matrices and bias vectors for each layer in the network. function systematically initializes these parameters based on the specified network architecture, which includes the input size, hidden layer sizes, and output size. The sizes of the weight matrices and bias vectors are determined by the dimensions of the layers they connect.

   Specifically, for each layer, a weight matrix is initialized with random values drawn from a standard normal distribution. The dimensions of this weight matrix correspond to the number of units in the previous layer and the current layer. This random initialization helps in breaking symmetry, which is essential for effective learning during training. Additionally, bias vectors are initialized as zero vectors, with dimensions matching the number of units in the current layer. This ensures that each neuron has an initial bias, which can be adjusted during training to better fit the data.

   The initialized parameters are stored in a dictionary, which maps each parameter (weights and biases) to its corresponding layer in the network. These initial parameters are crucial as they serve as the starting point for the optimization process. Subsequently, the best parameters are set to these initial values, providing a baseline from which the network can evolve during training. This methodical initialization lays a solid foundation for the network, promoting efficient convergence and enhancing the model's ability to learn complex patterns from the data.

3. **Activation Functions:**

   For activation functions, two primary activation functions have been utilized; the sigmoid function and the softmax function. The sigmoid function has been used for the hidden layers of neural networks which transforms input values into a range

between 0 and 1, introduces non-linearity to the model, while also enabling the network to learn and represent intricate patterns in the data.

On the other hand, the softmax function, where used in the output layer for classification tasks. It computes the exponential of each input value, normalizes these values by the sum of all exponentials, and produces a probability distribution over the output classes. This function is particularly useful for multi-class classification problems, as it ensures that the output probabilities sum to 100%, facilitating clear and interpretable classification decisions.

4. **Forward Propagation:**

   In forward propagation, the input data is passed through the network to generate predictions. In this implementation, forward propagation involves computing the activations of each layer sequentially. Starting with the input layer, the input data is multiplied by the weights and biases of the first layer. The resulting values are then passed through an activation function, such as the sigmoid function for hidden layers or the softmax function for the output layer. This process continues layer by layer, with each layer's output serving as the input for the next layer. The final output of the network is a set of predictions based on the transformed and processed input data. This method ensures that the network effectively captures and transforms the input features to make accurate predictions.

5. **Prediction:**

   The prediction process leverages the forward propagation mechanism. The input data is passed through the network, layer by layer, where each layer’s output serves as the input for the next. Activation functions, such as the sigmoid function for hidden layers and the softmax function for the output layer, are applied to transform the data at each step. The final layer produces the network’s output, which represents the model’s predictions. In classification tasks, the softmax function converts the outputs into probability distributions across different classes, and the class with the highest probability is selected as the predicted label.

6. **Population Initialization:**

   This function is tasked with generating an initial population of candidate solutions, each representing a potential set of parameters for the neural network. At the heart of this process lies the random initialization of individuals, where each individual encapsulates a distinct combination of weights and biases for the network's layers. The size of the population, dictated by the population size parameter, determines the number of individuals in the population. Leveraging a uniform distribution with specified lower and upper bounds, typically set at -0.3 and 0.3, the function creates a diverse set of initial solutions.

7. **Fitness Calculation:**

   The process of fitness calculation involves assessing the fitness or performance of each individual within the population. In the context of this implementation, fitness calculation entails evaluating the accuracy or effectiveness of each candidate solution in solving the underlying problem. Leveraging the training dataset, each individual's set of parameters is utilized to construct a neural network model. Subsequently, the model's predictions are compared against the ground truth labels to compute a fitness score, often based on a performance metric such as accuracy, precision, or loss function value. This fitness score quantifies the ability of the individual to accurately capture patterns and relationships within the data. Higher fitness scores indicate individuals who produce more desirable outcomes and are thus more likely to contribute positively to the evolutionary process. Through iterative evaluation and selection based on fitness, the genetic algorithm guides the evolution of the population towards increasingly optimal solutions, ultimately enhancing the neural network's performance and effectiveness in tackling the target task.

8. **Genome Reshaping:**

   This process involves transforming the genetic representation of individuals, typically encoded as vectors of parameters, to conform to the structure and requirements of the optimization algorithm. In the context of this implementation, genome reshaping primarily focuses on restructuring the genetic representation to align with the neural network's architecture. As each individual in the population encodes parameters

corresponding to the network's weights and biases, reshaping entails rearranging these parameters into the appropriate format for constructing the network. This may involve partitioning the parameter vector into weight matrices and bias vectors corresponding to each layer of the network, ensuring consistency with the network's architecture. Additionally, reshaping may encompass adjusting the dimensions and shapes of parameter matrices to match the specified layer sizes and dimensions. By harmonizing the genetic representation with the neural network's structure, genome reshaping enables the genetic algorithm to effectively operate on parameter sets, facilitating the evolutionary exploration and optimization of the solution space.

9. **Crossover:**

   This evolutionary mechanism emulates the concept of natural reproduction, where genetic material from two parent individuals is exchanged to produce offspring with diverse characteristics. In the context of this hybrid neural network framework, crossover serves as a mechanism for exploring the solution space by recombining parameters encoded within individuals' genomes. During crossover, pairs of parent individuals are selected based on their fitness or performance, typically favoring individuals with higher fitness scores. Subsequently, genetic material, represented as parameter vectors encoding weights and biases for the neural network, is exchanged between the selected parents to generate offspring individuals.

10. **Mutation:**

    The mutation functions begin by generating a binary mutation mask, wherein each element's probability of being mutated is determined by the specified mutation rate. Subsequently, random mutation changes are introduced within a predefined range, symmetrically distributed around zero. These mutation changes are then applied to the offspring individuals, selectively based on the mutation mask, effectively perturbing their genetic material. Finally, the mutated offspring, incorporating both stochastic changes and genetic inheritance from the parents, are returned as the output of the mutation process. This adaptive mutation mechanism fosters population diversity, enabling exploration of the solution space by introducing controlled stochastic variations to the offspring's genetic makeup.

11. Optimization Loop:

The "optimize" function orchestrates the optimization process that drives the genetic algorithm integrated with the neural network model. The initial step involves generating a population of potential solutions through the population initialization function. Among these solutions, the genetic representation of the current best-performing model parameters, referred to as the "best genome," is reshaped into a genome-like vector and incorporated into the initial population, thus ensuring that the best-known solution is always part of the evolutionary process.

The optimization proceeds over a series of generations, each iteratively refining the population. During each generation, the function evaluates the fitness of each individual within the population by assessing their performance on the provided training dataset. This evaluation is based on fitness scores, which reflect the accuracy of each individual's predictions compared to the actual outcomes.

Key steps in each generation include:

1. **Evaluation of Fitness**: The fitness scores for all individuals in the population are computed, ranking the individuals based on their accuracy.
2. **Selection of Elites**: The top-performing individuals, or elites, are identified. These elite individuals, determined by the highest fitness scores, are retained for the subsequent generation to ensure that the best solutions are not lost during the evolutionary process.
3. **Tracking the Best Solution**: The algorithm continuously monitors the individual with the highest accuracy across all generations. If an individual's accuracy surpasses the current best accuracy, the model's parameters are updated to reflect this new optimal solution.
4. **Generation of New Individuals**: New candidate solutions are generated through two primary genetic operations:
    - **Crossover**: This operation allows for the recombination of genetic material from pairs of elite individuals, promoting the exploration of new potential solutions by combining successful traits from different elites.

- **Adaptive Mutation**: This process introduces controlled random changes to the offspring, enhancing the algorithm's ability to explore the solution space and avoid local optima.

The newly generated individuals from crossover and mutation are then combined with the elite individuals to form the next generation's population. This ensures that the population continually evolves and explores new regions of the solution space, while retaining the best solutions discovered so far.

At each generation, the algorithm outputs the best accuracy achieved, providing insight into the progression of the optimization process. This iterative approach of evaluation, selection, crossover, and mutation leads to the refinement and evolution of the population towards an optimal solution for the given task.

Upon completion of the specified number of generations, the function returns the best parameters identified throughout the optimization process. These parameters represent the most accurate and robust configuration of the neural network model for the target task, derived through the synergistic application of genetic algorithm principles and neural network learning.

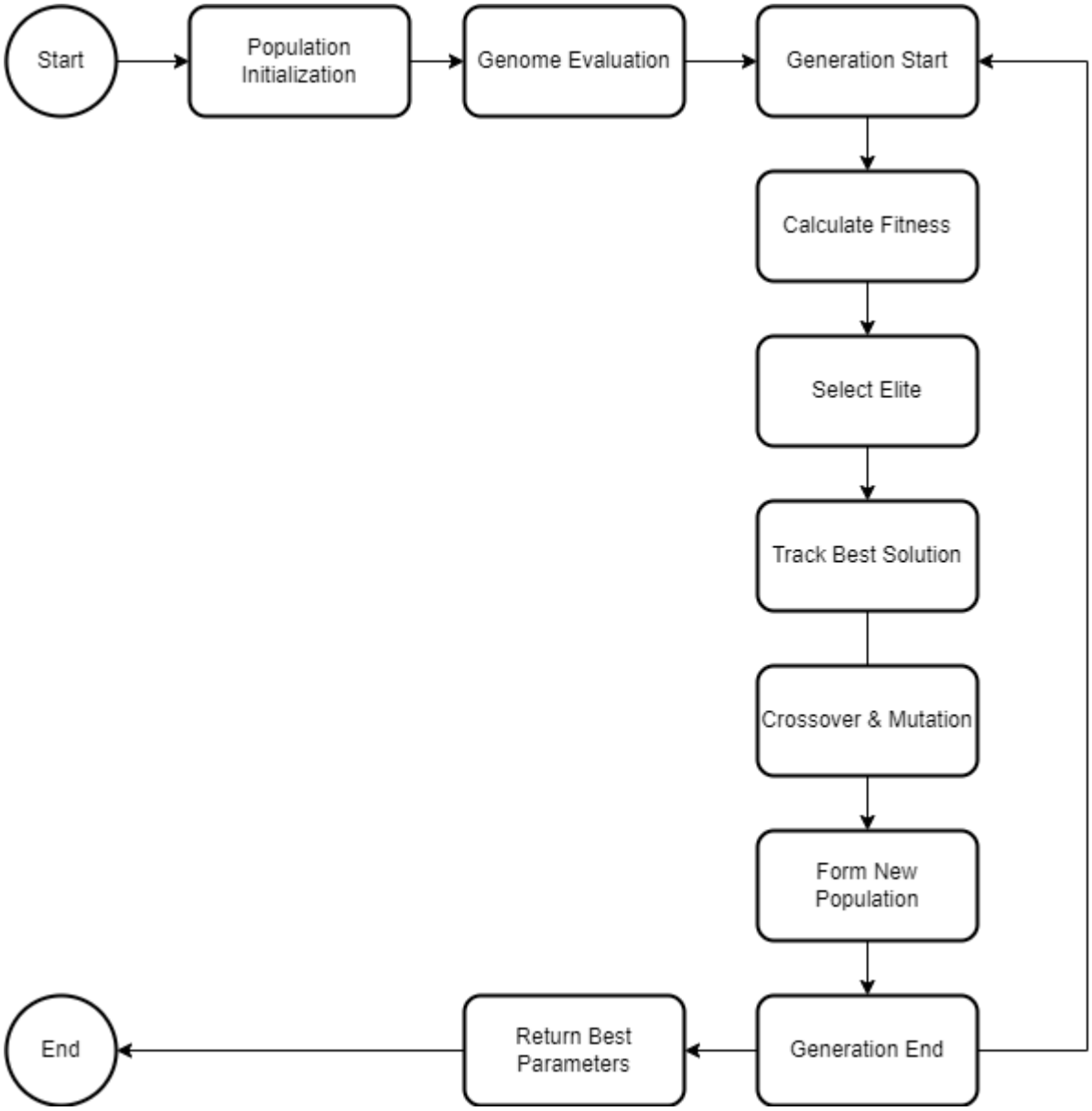


Figure 5: Optimization Loop

Several computationally intensive steps shape the time complexity of the "optimize" function in the genetic algorithm integrated with the neural network model. The initial population generation, which involves creating potential solutions, has a complexity of O(P×N), where P is the population size and N is the number of model parameters. Each generation evaluates individuals' fitness by training the neural network on a dataset of size M, resulting in a complexity of O(P×M×N) per generation. The selection of elite individuals requires sorting the population, adding a complexity of O(P log P). Generating new individuals through crossover and adaptive mutation contributes an additional O(P×N) per generation. The process of tracking the best solution adds a further O(P) complexity. Summing these

complexities across all generations G, the overall time complexity of the optimization loop is $O\big(G \times (P \times M \times N + P \log P + P \times N)\big)$. This high complexity underscores the significant computational resources required to achieve an optimized solution, particularly in scenarios involving large populations, extensive parameter sets, and multiple generations.

Training:

The train function is responsible for training the neural network model, incorporating both traditional backpropagation and GA optimization to refine the model's parameters. The function takes the following steps:

1. **One-Hot Encoding of Labels:**

   The function begins by encoding the target labels (train and test) into one-hot format. This encoding is necessary for the network to compute the loss during training, especially when dealing with multi-class classification problems.

2. **Training Loop (Epochs):**

   For each epoch in the training loop:

   - **Forward Propagation and Loss Calculation:** During forward propagation, the model processes the training data using the forward propagation method to compute predicted probabilities for each class, which are stored in the cache. The loss for the training data is then calculated using cross-entropy loss, focusing on the probabilities of the true labels. To evaluate performance, the accuracy of the training set is computed by comparing the predicted classes from the output layer with the actual labels using the accuracy score metric.
   - **Backpropagation:** The backward propagation method is called to update the model's parameters based on the computed gradients. This step fine-tunes the parameters through traditional backpropagation before applying GA optimization.
   - **Tracking the Best Model:** If the current training accuracy exceeds the best accuracy, the model updates the best accuracy and best loss. The parameters dictionary is also updated to store the current best model parameters.
   - **Genetic Algorithm Optimization:** After backpropagation, the optimize method is invoked to further optimize the parameters using a genetic algorithm. This

step introduces a global search mechanism, potentially finding better solutions that might be missed by gradient-based optimization alone.

- **Prediction and Evaluation:** The model makes predictions on both the training and testing sets using the predict method. The training and testing accuracy is then calculated by comparing the predictions with the true labels. These accuracy scores provide insight into the model's performance on both seen (training) and unseen (testing) data.

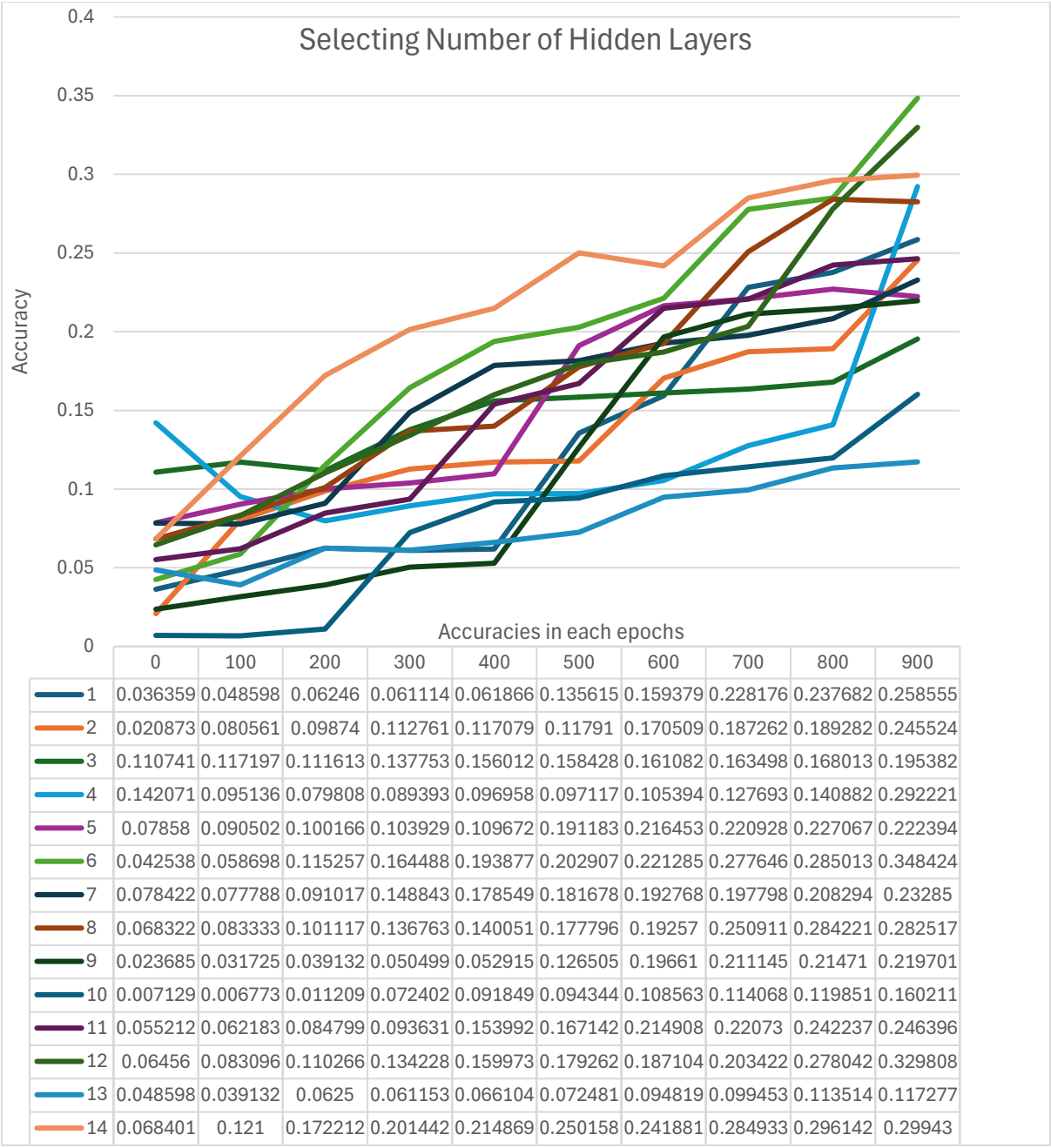
3.3.4. Algorithm Parameters

The parameters for both the neural network (NN) and the GA are defined. These parameters are crucial for configuring the architecture and behavior of the algorithms used in the model. The neural network parameters consist of (Input, Hidden layers, Neurons, Output, and Learning rate). On the other hand, the genetic algorithm parameters consist of (Genome, Population, Elite, Mutation, and Generation).

Neural Network Parameters:

1- **Input Size**: The input size has been selected based on the number of features after applying the PCA which is 14.

2- **Hidden Layers**: The number of hidden layers was selected based on the following experimentation. The algorithm was trained 14 times, each time increasing the number of hidden layers and running for 1000 epochs. After thorough analysis, it was observed that all trials followed the same trend as shown in the below figure.

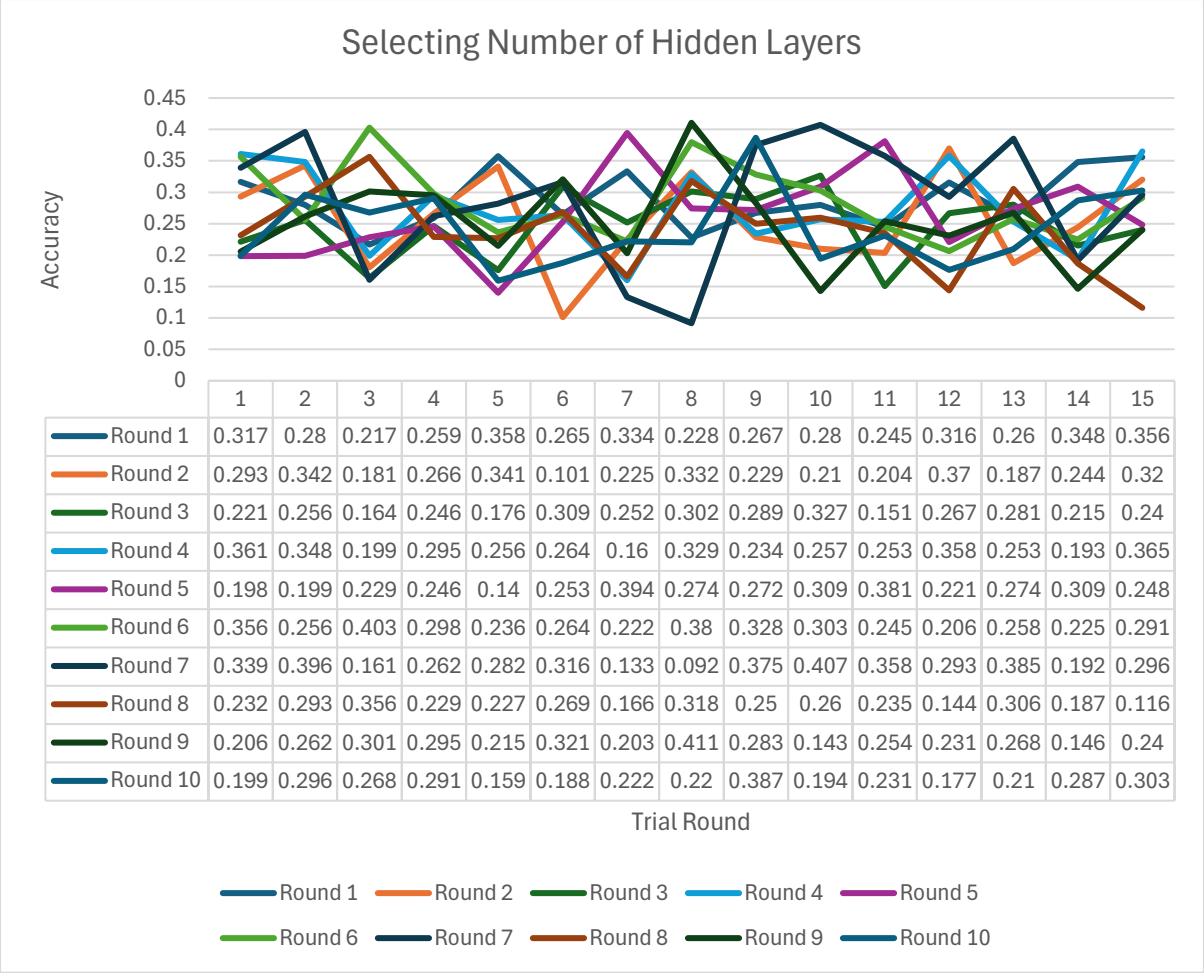


Accuracies in each epochs

	0	100	200	300	400	500	600	700	800	900
1	0.036359	0.048598	0.06246	0.061114	0.061866	0.135615	0.159379	0.228176	0.237682	0.258555
2	0.020873	0.080561	0.09874	0.112761	0.117079	0.11791	0.170509	0.187262	0.189282	0.245524
3	0.110741	0.117197	0.111613	0.137753	0.156012	0.158428	0.161082	0.163498	0.168013	0.195382
4	0.142071	0.095136	0.079808	0.089393	0.096958	0.097117	0.105394	0.127693	0.140882	0.292221
5	0.07858	0.090502	0.100166	0.103929	0.109672	0.191183	0.216453	0.220928	0.227067	0.222394
6	0.042538	0.058698	0.115257	0.164488	0.193877	0.202907	0.221285	0.277646	0.285013	0.348424
7	0.078422	0.077788	0.091017	0.148843	0.178549	0.181678	0.192768	0.197798	0.208294	0.23285
8	0.068322	0.083333	0.101117	0.136763	0.140051	0.177796	0.19257	0.250911	0.284221	0.282517
9	0.023685	0.031725	0.039132	0.050499	0.052915	0.126505	0.19661	0.211145	0.21471	0.219701
10	0.007129	0.006773	0.011209	0.072402	0.091849	0.094344	0.108563	0.114068	0.119851	0.160211
11	0.055212	0.062183	0.084799	0.093631	0.153992	0.167142	0.214908	0.22073	0.242237	0.246396
12	0.06456	0.083096	0.110266	0.134228	0.159973	0.179262	0.187104	0.203422	0.278042	0.329808
13	0.048598	0.039132	0.0625	0.061153	0.066104	0.072481	0.094819	0.099453	0.113514	0.117277
14	0.068401	0.121	0.172212	0.201442	0.214869	0.250158	0.241881	0.284933	0.296142	0.29943

Figure 6: Selecting Number of Hidden Layers 1

After conducting the initial experimentation, the process was repeated 10 times, with the performance of the model evaluated each time. For each trial, the average performance for different numbers of hidden layers was calculated. Subsequently, the configuration with two hidden layers, which provided the best average performance, was selected as illustrated in the figure below and highlighted in bold in the accompanying table.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Round 1	0.317	0.28	0.217	0.259	0.358	0.265	0.334	0.228	0.267	0.28	0.245	0.316	0.26	0.348	0.356
Round 2	0.293	0.342	0.181	0.266	0.341	0.101	0.225	0.332	0.229	0.21	0.204	0.37	0.187	0.244	0.32
Round 3	0.221	0.256	0.164	0.246	0.176	0.309	0.252	0.302	0.289	0.327	0.151	0.267	0.281	0.215	0.24
Round 4	0.361	0.348	0.199	0.295	0.256	0.264	0.16	0.329	0.234	0.257	0.253	0.358	0.253	0.193	0.365
Round 5	0.198	0.199	0.229	0.246	0.14	0.253	0.394	0.274	0.272	0.309	0.381	0.221	0.274	0.309	0.248
Round 6	0.356	0.256	0.403	0.298	0.236	0.264	0.222	0.38	0.328	0.303	0.245	0.206	0.258	0.225	0.291
Round 7	0.339	0.396	0.161	0.262	0.282	0.316	0.133	0.092	0.375	0.407	0.358	0.293	0.385	0.192	0.296
Round 8	0.232	0.293	0.356	0.229	0.227	0.269	0.166	0.318	0.25	0.26	0.235	0.144	0.306	0.187	0.116
Round 9	0.206	0.262	0.301	0.295	0.215	0.321	0.203	0.411	0.283	0.143	0.254	0.231	0.268	0.146	0.24
Round 10	0.199	0.296	0.268	0.291	0.159	0.188	0.222	0.22	0.387	0.194	0.231	0.177	0.21	0.287	0.303

Figure 7: Selecting Number of Hidden Layers 2

Table 8: Best Number of Hidden Layers

Hidden Layers	Round 1	Round 2	Round 3	Round 4	Round 5	Round 6	Round 7	Round 8	Round 9	Round 10	Average
1	0.3167	0.2929	0.2214	0.3608	0.1982	0.3561	0.3388	0.2316	0.2056	0.199	0.27211
2	0.2799	0.3424	0.256	0.3482	0.1987	0.2558	0.3958	0.2929	0.2616	0.2961	**0.29274**
3	0.2168	0.1806	0.1639	0.1987	0.2286	**0.4029**	0.1605	**0.3564**	0.3013	0.2675	0.24772
4	0.2588	0.2663	0.2464	0.2947	0.2458	0.2978	0.2623	0.2288	0.2952	0.2906	0.26867

5	**0.3575**	0.3411	0.1761	0.2559	0.1401	0.2363	0.2818	0.2268	0.2149	0.1592	0.23897
6	0.2648	0.1012	0.3087	0.2637	0.2529	0.2643	0.3161	0.2689	0.3207	0.1879	0.25492
7	0.3335	0.225	0.2517	0.1601	**0.3944**	0.2218	0.1331	0.1656	0.2031	0.2218	0.23101
8	0.2278	0.332	0.3015	0.3294	0.2742	0.3797	0.0916	0.3183	**0.4106**	0.2203	0.28854
9	0.2673	0.2287	0.289	0.2341	0.2715	0.3281	0.3754	0.2496	0.2832	**0.3868**	0.29137
10	0.2799	0.2102	**0.3267**	0.2569	0.3088	0.303	**0.4073**	0.2596	0.1428	0.1942	0.26894
11	0.245	0.2037	0.1507	0.2528	0.3811	0.2451	0.3578	0.2353	0.254	0.2312	0.25567
12	0.3157	**0.3698**	0.2666	0.358	0.2207	0.2064	0.2928	0.1438	0.2309	0.1767	0.25814
13	0.2601	0.1871	0.2806	0.2534	0.274	0.2577	0.3853	0.3056	0.2679	0.2095	0.26812
14	0.348	0.2444	0.2151	0.1927	0.3089	0.2248	0.192	0.1869	0.1462	0.2868	0.23458
15	0.3559	0.3202	0.2403	**0.365**	0.2483	0.2912	0.2961	0.1161	0.2404	0.3026	0.27761

3- **Neurons Layers**: Determining the optimal number of neurons in hidden layers remains a challenge in neural network design, as there is no definitive solution applicable to all scenarios. Various studies in the field of neural networks acknowledge this challenge and propose heuristic methods for selecting the number of neurons [98], [99], [100], [101]. The heuristic method employed in this study involves setting the number of neurons in the first hidden layer to two-thirds of the total number of input and output neurons, followed by using the full size for the second hidden layer. Specifically, for an

input size of 14 neurons and an output size of 15 neurons, the hidden layers were configured with 19 neurons in the first layer and 29 neurons in the second layer.

4- **Learning rate:** The learning rate has been set to 0.1 based on several related papers where the authors always select 0.1 as a default initial step for their proposal [102], [103], [104].
5- **Output Size:** has been selected based on the number of classes which is 15.
6- **Epochs**: has been set to 1000, considering the computational power.

Genetic Algorithm Parameters:

1- **Genome Size**: The genome size was determined to be 1,315. This value reflects the total number of weights and biases in the neural network and was calculated by summing all the parameters between the input layer, the hidden layers, and the output layer. Specifically, the network architecture includes an input layer with 14 neurons, two hidden layers with 19 and 29 neurons respectively, and an output layer with 15 neurons. The genome size calculation takes into account the connections and biases from the input layer to the first hidden layer, between the hidden layers, and from the last hidden layer to the output layer. This comprehensive calculation ensures that the genome fully encapsulates the neural network's structure, allowing the genetic algorithm to effectively optimize all parameters. The chosen value of 1,315 strikes a balance between adequately representing the network's complexity and maintaining manageable computational demands, thereby facilitating efficient and effective training.
2- **Population Size**: The population size is a critical parameter in genetic algorithms as it determines the number of candidate solutions available in each generation [105]. To determine the optimal population size for this study, an experiment was conducted by varying the population size from 10% to 100% of the genome size in 10% increments. The population sizes tested ranged from 131 to 1,315, and the corresponding test accuracies were recorded to assess selecting the best population size.

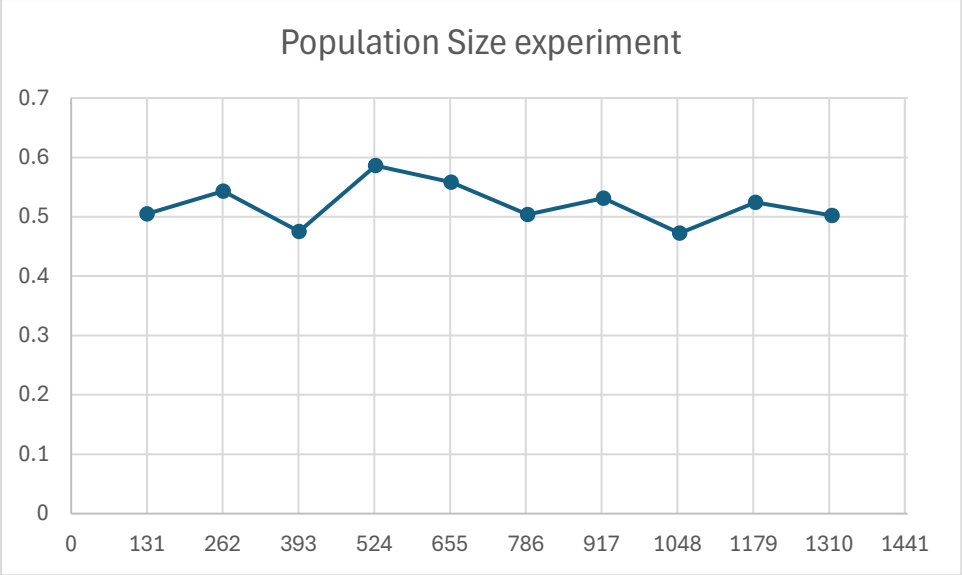


Figure 8: Population Size Experiment

Table 9: Population Size Experiment

Population Size	Test Accuracy
131	0.504753
263	0.543251
394	0.474968
526	0.586027
657	0.55846
789	0.503802
920	0.53121
1052	0.472275
1183	0.52446
1315	0.50206

The above table illustrates how test accuracy varies with different population sizes. Notably, all the results hover around 50%. Even though the highest test accuracy of 0.586027 was achieved with a population size of 526, the population size has been set to 1315 since all the test results are in the same range. Additionally, increasing the population size allows for more extensive exploration, potentially leading to better overall optimization.

3- **Elite Size**: The elite size, which determines the proportion of the top-performing individuals retained for the next generation, was set to 20% of the population size. For a population size of 1315, this corresponds to 263 individuals. The selection of 20% as the elite size is aimed at maintaining a balance between preserving the best solutions and introducing new genetic material into the population. By retaining the top 20%, the algorithm ensures that high-quality solutions are carried forward, providing a stable foundation for further optimization while still allowing for a significant degree of exploration and variation among the rest of the population.
4- **Mutation Rate**: The mutation rate was set to 0.3, which means that 30% of the genes in each individual have a chance of being altered during each generation. Mutation introduces random variations in the population, which is crucial for maintaining genetic diversity and allowing the algorithm to explore new areas of the solution space that might not be reached through crossover alone.
5- **Number of Generations**: In this study, 1,000 generations were found to be a suitable balance, allowing enough evolutionary cycles to optimize the neural network parameters effectively without excessive computational burden. This extensive evolutionary process helps to ensure that the algorithm thoroughly explores the solution space and converges towards a highly accurate and reliable set of parameters for the neural network.

4- Results and Analysis

The Results and Analysis section marks the effectiveness of the developed hybrid algorithm for cyber-attack detection in the IIoT landscape. This comprehensive evaluation, aims to unveil insights derived from applying the GA optimization strategy, shedding light on the performance of the developed model. The analytical journey involves a detailed review of the classification report, confusion matrix, and ROC curve metrics for all types of cyber-attacks and normal traffic.

After implementing the algorithm detailed in Section 3.3 (Implementation), the neural network was configured with the parameters listed in the table below. These parameters reflect the final state of the neural network architecture.

Table 10: The Final Neural Network Architecture

Parameter Name	Parameter Shape	Number of Parameters
Weight 1	(14, 19)	266
Bias 1	(1, 19)	19
Weight 2	(19, 29)	551
Bias 2	(1, 29)	29
Weight 3	(29, 15)	435
Bias 3	(1, 15)	15

Moreover, to ensure the robustness and reliability of the results, the algorithm was trained three times with different stratified sampling training datasets. This method was employed to generate varied samples in each run, thereby reducing the potential impact of any single random initialization on the results. This iterative approach was employed to verify the consistency of the outcomes and to confirm that the results were not a product of random variation or overfitting.

4.1. Classification Metrics

In the context of evaluating classification models, it is essential to understand key metrics derived from a confusion matrix. A confusion matrix is a table used to describe the performance of a classification model by comparing the predicted labels with the true labels. It includes four main metrics [106]:

Table 11: Confusion Matrix Description

Metrics	Description
True Positive (TP)	An instance where both predicted and actual values are positive
False Positive (FP)	An instance where the predicted value is positive, but the actual value is negative
False Negative (FN)	An instance where the predicted value is negative, but the actual value is positive
True Negative (TN)	An instance where both the predicted and actual values are negative

Based on these, several classification metrics are commonly used to evaluate the model's performance [106], [107], [108]:

Precision measures the accuracy of positive predictions. It is defined as the ratio of true positive predictions to the total number of positive predictions made by the model. A high precision indicates when the model predicts a positive class, it is likely to be correct. The formula for precision is:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall also known as sensitivity or true positive rate, it assesses the model's ability to capture all relevant positive instances. It is the ratio of true positive predictions to the total number of actual positive instances in the dataset. High recall signifies that the model is effective in identifying most of the positive instances. The formula for the recall is:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

F1-Score provides a balanced measure of a model's performance by combining precision and recall into a single metric. It is particularly useful when there is an imbalance between positive and negative classes. The F1-Score is the harmonic mean of precision and recall. The formula for F1-Score is:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy measures the overall correctness of the model's predictions. It is the ratio of correctly predicted instances (both true positives and true negatives) to the total number of instances. Accuracy is a general measure of model performance, but it can be misleading if the classes are imbalanced. The formula for accuracy is:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total Number of Instances}}$$

Support refers to the number of actual occurrences of each class in the dataset. It is used to provide context to the precision, recall, and F1-Score metrics by showing the distribution of true instances across different classes.

The following tables present the detailed classification reports for three different stratified sampling training datasets. These tables highlight the model's performance in various attack types and normal traffic to evaluate the model's effectiveness, key classification metrics, including precision, recall, F1-score, support, and accuracy, were utilized.

Table 12: Classification Report – First Run

Class	Precision	Recall	F1-score	Support
Backdoor	0.96	0.88	0.92	411
DDoS_HTTP	1	0.99	0.99	416
DDoS_ICMP	0.99	1	0.99	579
DDoS_TCP	1	1	1	416
DDoS_UDP	1	0.99	1	550
Fingerprinting	0	0	0	38
MITM	0	0	0	49
Normal	0.99	1	0.99	1003
Password	0.99	0.96	0.97	387
Port_Scanning	0.99	1	0.99	365
Ransomware	0.85	1	0.92	492
SQL_injection	0.99	0.67	0.8	414
Uploading	0.74	0.99	0.85	400
Vulnerability_scanner	0.94	0.97	0.95	386
XSS	0.87	0.92	0.9	406
Accuracy			0.94	6312

Table 13: Classification Report – Second Run

Class	Precision	Recall	F1-score	Support
Backdoor	0.96	0.88	0.92	411
DDoS_HTTP	1	0.99	0.99	416
DDoS_ICMP	1	1	1	579
DDoS_TCP	1	1	1	416
DDoS_UDP	1	1	1	550
Fingerprinting	0	0	0	38
MITM	0	0	0	49
Normal	0.99	1	1	1003
Password	0.99	0.98	0.98	387
Port_Scanning	0.99	1	0.99	365
Ransomware	0.85	1	0.92	492
SQL_injection	0.99	0.67	0.8	414
Uploading	0.74	0.99	0.85	400
Vulnerability_scanner	0.96	0.97	0.96	386
XSS	0.87	0.92	0.9	406
Accuracy			0.95	6312

Table 14: Classification Report – Third Run

Class	Precision	Recall	F1-score	Support
Backdoor	0.95	0.87	0.91	411
DDoS_HTTP	1	0.99	1	416
DDoS_ICMP	1	1	1	579
DDoS_TCP	1	1	1	416
DDoS_UDP	1	1	1	550
Fingerprinting	0	0	0	38
MITM	0	0	0	49
Normal	0.99	1	1	1003
Password	0.99	0.97	0.98	387
Port_Scanning	1	1	1	365

Ransomware	0.46	1	0.63	492
SQL_injection	0.99	0.69	0.81	414
Uploading	0.80	0.34	0.47	400
Vulnerability_scanner	0.97	0.92	0.94	386
XSS	0.82	0.94	0.87	406
Accuracy			0.90	6312

The following analysis presents an overview of the algorithm's performance in classifying various cyber-attack types and normal network traffic, based on three different runs. Precision metrics reflect the accuracy of positive predictions, with notable performances observed in several attack categories:

- **DDoS ICMP, DDoS TCP, and DDoS UDP attacks** demonstrated perfect precision and recall across all runs, indicating flawless detection capabilities. This suggests that the algorithm is highly effective in identifying these specific types of DDoS attacks.
- **DDoS HTTP** also exhibited near-perfect precision and recall, reinforcing the algorithm's robustness in detecting this attack vector.
- **Normal traffic** consistently achieved high precision and recall, confirming the model's strong ability to distinguish benign traffic from malicious activities.

Conversely, certain classes, such as **Fingerprinting** and **MITM**, consistently showed a lack of detection capability, reflected in zero precision, recall, and F1-scores. This indicates that the algorithm struggled to identify these attack types across all runs.

Ransomware exhibited varied performance. While its recall was high (1.00), its precision was lower especially in the third run, leading to a lower F1-score overall. This variation suggests challenges in achieving a balance between detecting and correctly classifying ransomware instances.

SQL Injection showed high precision (0.99) but lower recall with in all runs achieving the highest (0.69) in the third run, indicating that while the algorithm is effective at identifying SQL injection attacks when they are detected, it may miss a significant portion of such instances.

Uploading also demonstrated varied results, with lower recall in some runs, which might reflect challenges in detecting all instances of this attack type.

Overall, the algorithm achieved an **accuracy of approximately 90-95%** across the three runs, indicating strong performance. The weighted metrics further validate the model's effectiveness, with weighted average precision, recall, and F1-score ranging from 0.80 to 0.94, showcasing a balanced performance across the diverse class distribution. These results highlight the model's overall capability in classifying cyber-attacks, though certain areas, such as Fingerprinting and MITM detection, warrant further improvement.

Table 15: Classification Results - Average

Class	Precision	Recall	F1-score	Support
Backdoor	0.956667	0.876667	0.916667	411
DDoS_HTTP	1	0.99	0.993333	416
DDoS_ICMP	0.996667	1	0.996667	579
DDoS_TCP	1	1	1	416
DDoS_UDP	1	0.996667	1	550
Fingerprinting	0	0	0	38
MITM	0	0	0	49
Normal	0.99	1	0.996667	1003
Password	0.99	0.97	0.976667	387
Port_Scanning	0.993333	1	0.993333	365
Ransomware	0.72	1	0.823333	492
SQL_injection	0.99	0.676667	0.803333	414
Uploading	0.76	0.773333	0.723333	400
Vulnerability_scanner	0.956667	0.953333	0.95	386
XSS	0.853333	0.926667	0.89	406
Accuracy			**0.93**	6312

4.2. Confusion Matrix

The confusion matrix provides a detailed breakdown of the algorithm's performance by illustrating the true positives, false positives, true negatives, and false negatives for each class. In this matrix, the rows represent the actual classes, while the columns represent the predicted classes. Each cell in the matrix shows the number of instances for the corresponding actual and predicted class pair.

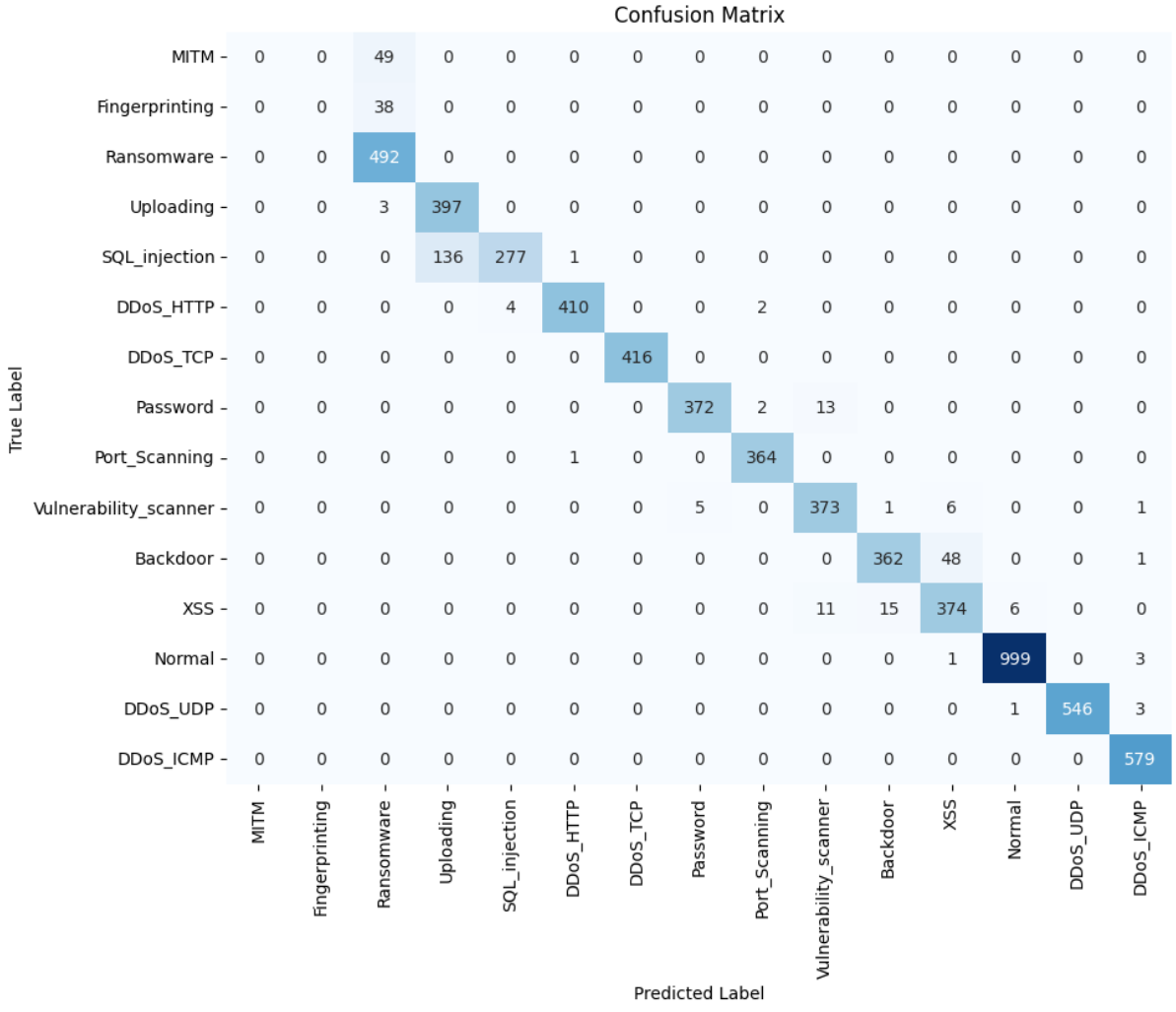


Figure 9: Confusion Matrix – First Run

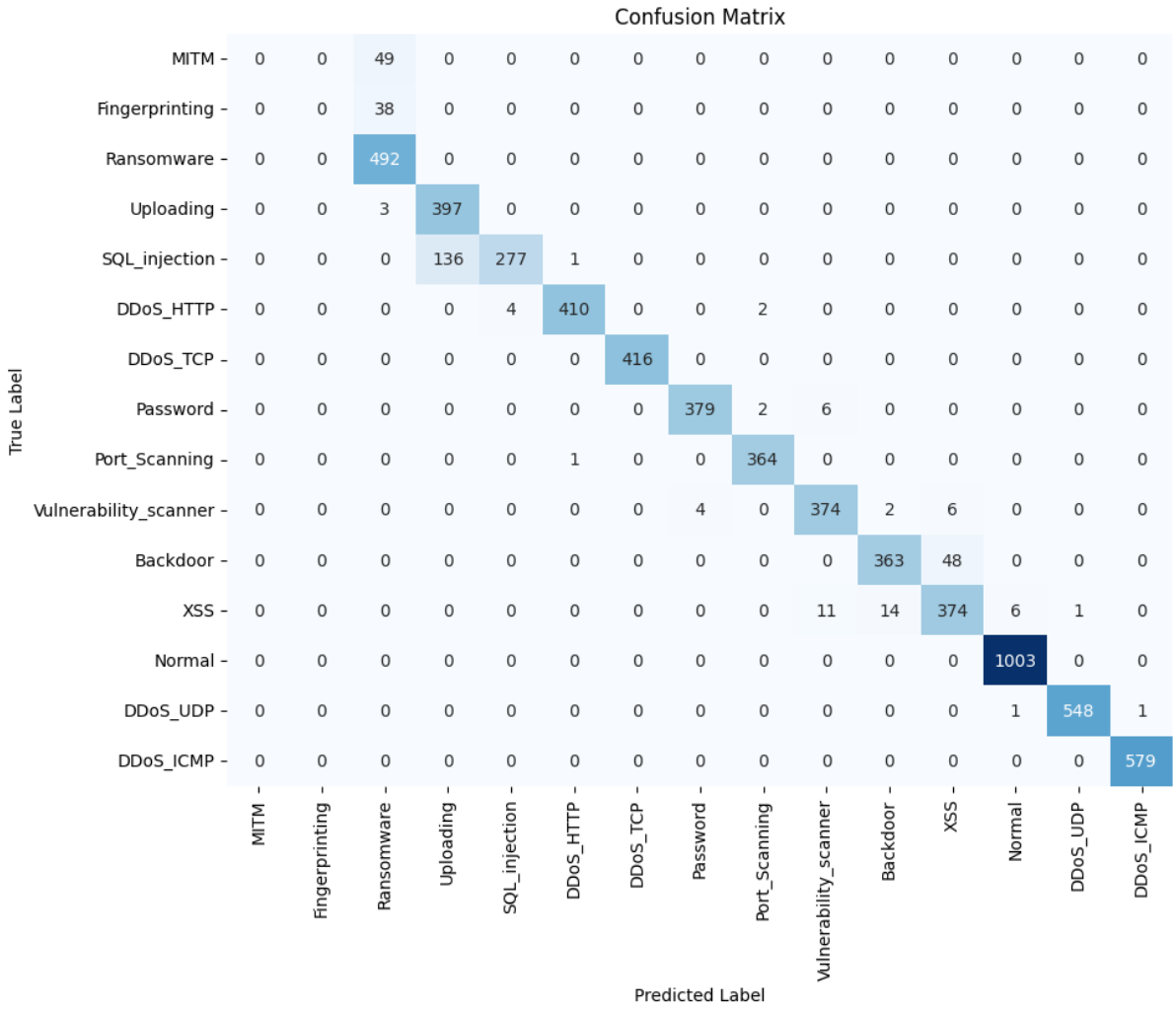


Figure 10: Confusion Matrix – Second Run

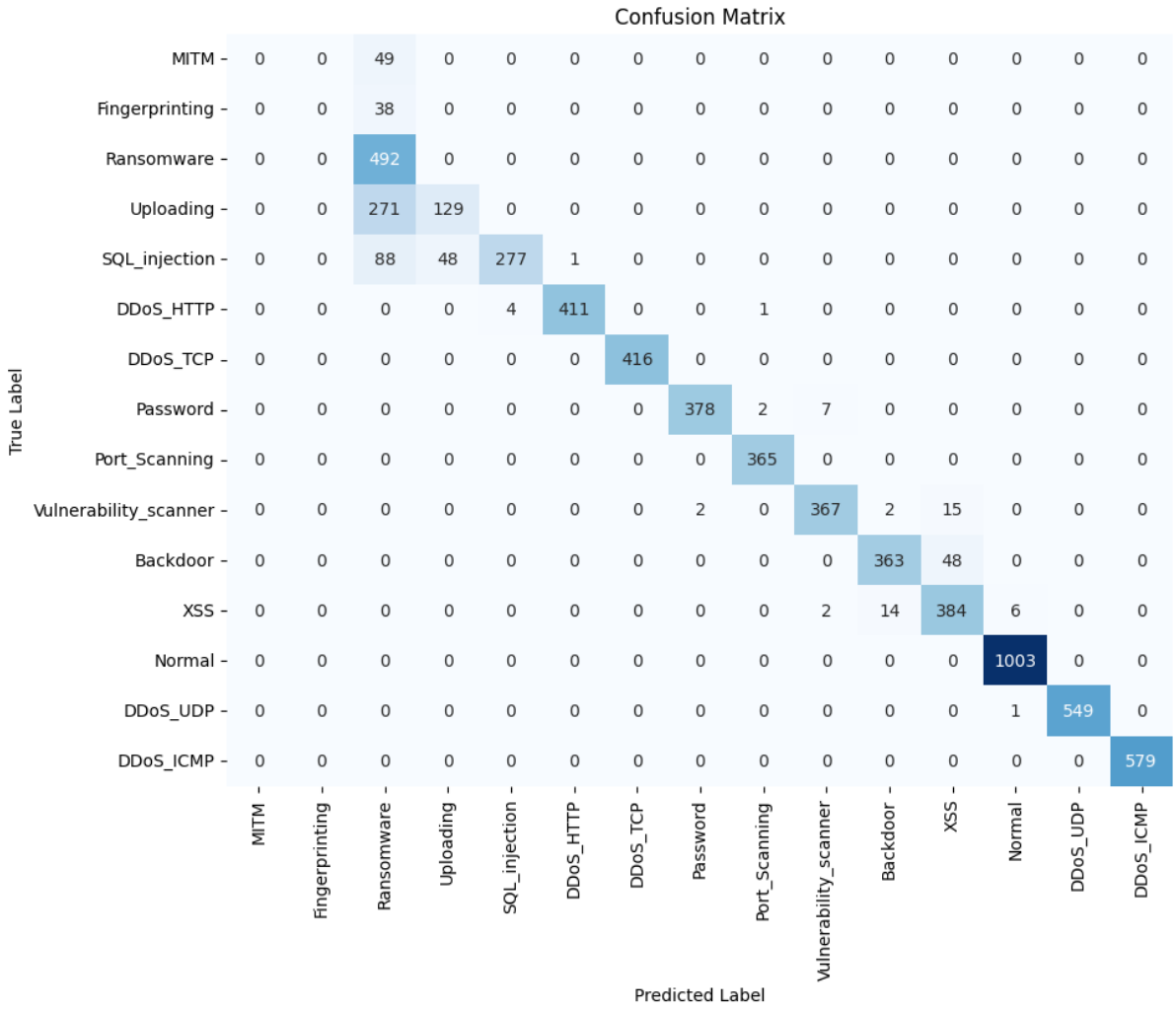


Figure 11: Confusion Matrix – Third Run

The confusion matrices presented in the figures above illustrate the performance of the classification model across different categories of cyber-attacks and normal traffic.

General Observations Across Runs:

1. Consistency in Classification:
    - **Ransomware, DDoS_TCP, DDoS_UDP, DDoS_ICMP, and Normal traffic** consistently show high accuracy across all three runs, indicating that the algorithm reliably detects these classes, regardless of the random state used.
    - The true positive rates for these categories are consistently high, showing the robustness of the model in identifying these types of traffic.
2. Variability in Certain Categories:
    - **SQL Injection, Uploading, and XSS** show variability across different random states. For instance, the classification accuracy for SQL Injection improves in

the third run compared to the first two runs, suggesting that the model's performance in these categories is sensitive to the initial conditions set by the random state.

- **Uploading and XSS** continue to show some level of misclassification in all runs, though with varying degrees of accuracy.

3. Impact of Random State:
    - The differences in the confusion matrices suggest that while the algorithm's overall performance is stable, the choice of random state can influence its effectiveness in distinguishing between more challenging categories, such as SQL Injection and Uploading.
    - For example, in the run with the second run, the model's accuracy in classifying Normal traffic increases, whereas, with the first run, there is a slight drop in the accuracy of certain categories.

The results from the three runs indicate that while the algorithm is generally robust, its performance can be influenced by the random state, particularly in more challenging categories. This suggests that further optimization, possibly by incorporating an ensemble approach or fine-tuning hyperparameters, could help to mitigate the variability and improve the overall accuracy of the model across all attack categories. This discussion highlights the importance of considering random state variability in model evaluation, especially when dealing with complex, multi-class classification tasks like cyber-attack detection.

4.3. ROC Curve

The Receiver Operating Characteristic (ROC) curve is a graphical representation commonly used to evaluate the performance of the classification model [109]. It illustrates the trade-off between the recall and the false positive rate as the discrimination threshold for classifying positive instances is varied.

The ROC curve is created by plotting the True Positive Rate (TPR), also known as sensitivity; same as the recall from the confusion matrix, against the False Positive Rate (FPR) at various threshold settings. Each point on the curve represents a pair of TPR and FPR values

corresponding to a specific decision threshold. A model with strong predictive performance will have a ROC curve that closely hugs the upper-left corner of the plot, indicating high sensitivity (few false negatives) and a low false positive rate [110]. The formula for FPR is:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

In addition to the ROC curve, the area under the ROC curve (AUC-ROC) is often calculated. AUC-ROC provides a single value summarizing the model's performance. A higher AUC-ROC value (closer to 1) suggests a better overall discriminatory ability of the model across different threshold settings.

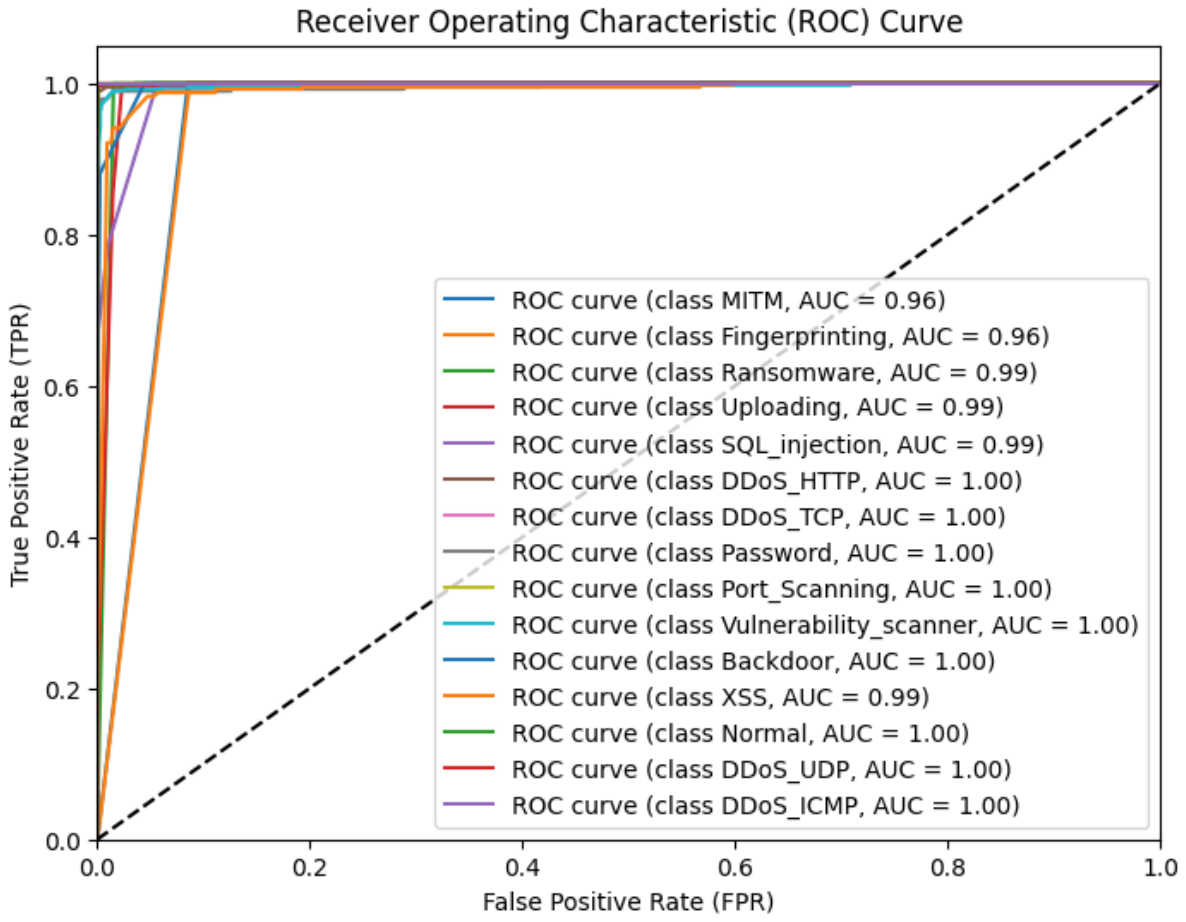


Figure 12: ROC Curve – First Run

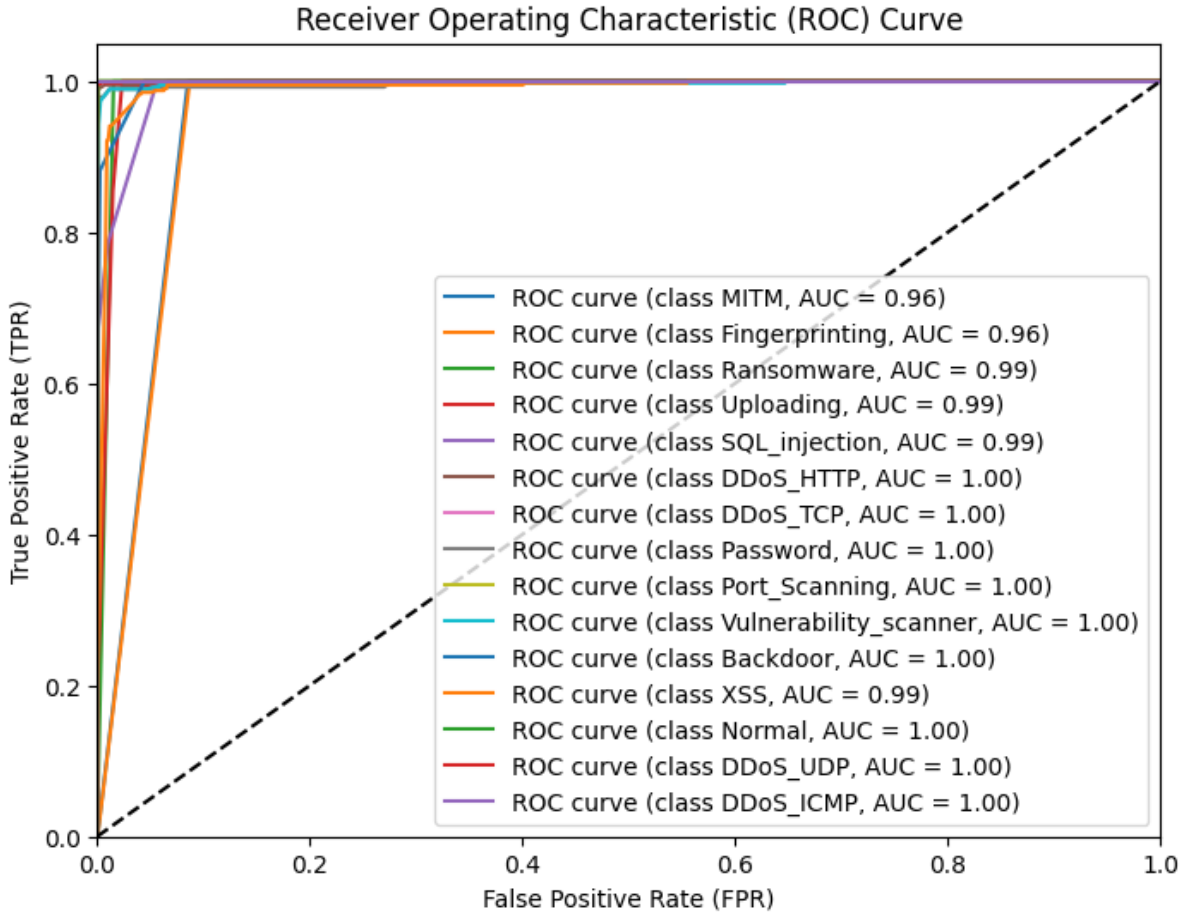


Figure 13: ROC Curve – Second Run

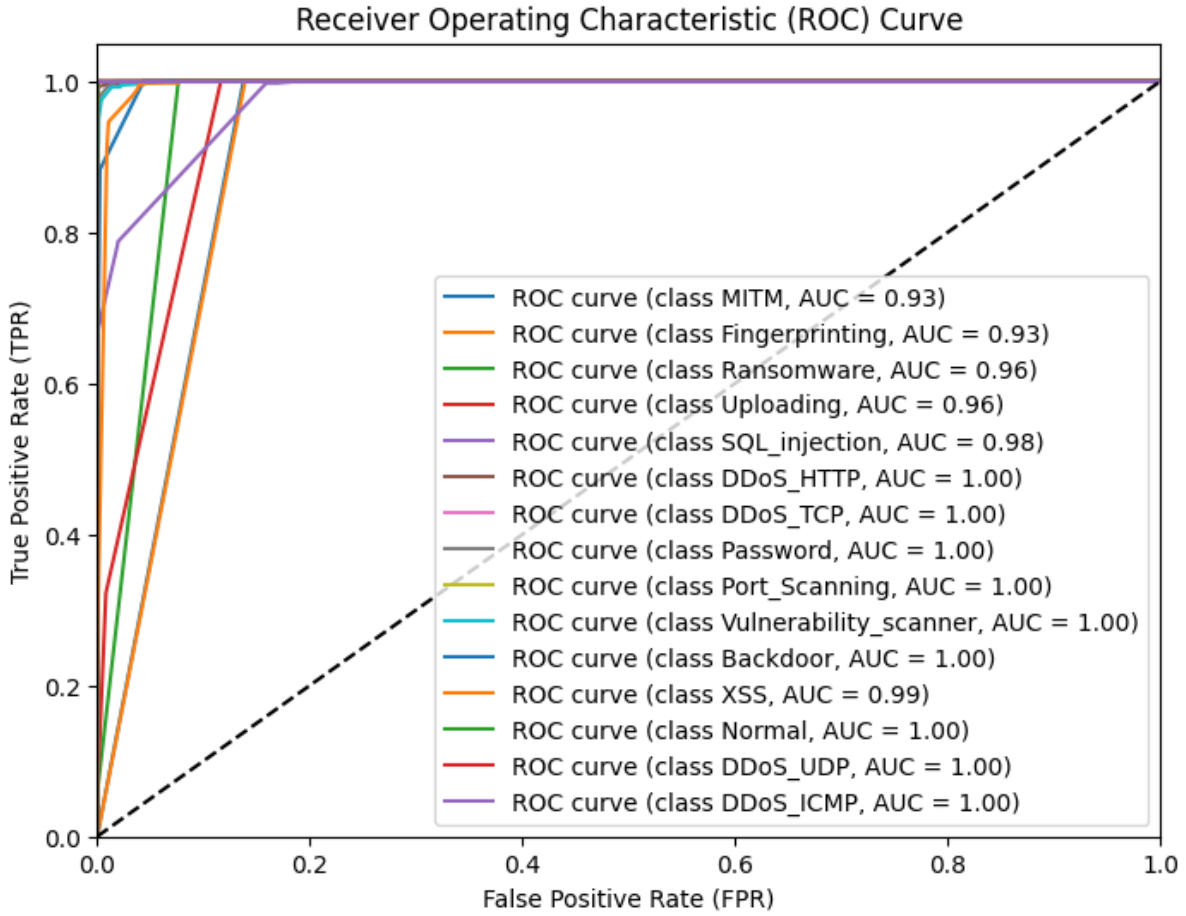


Figure 14: ROC Curve - Third Run

Notably, most classes consistently exhibit outstanding AUC values close to 1.0 across all runs, indicating robust and near-perfect discrimination capabilities. The DDoS_HTTP, DDoS_TCP, DDoS_UDP, DDoS_ICMP, Password, Port_Scanning, Vulnerability_scanner, Backdoor, and Normal classes consistently achieve an AUC of 1.00 in each run. This consistency underscores the classifier's exceptional performance in distinguishing these types of network traffic.

However, some classes show slight variability in AUC values across the different runs, though they remain high. The MITM (Man-in-the-Middle) and Fingerprinting classes, for instance, have AUC values ranging from 0.93 to 0.96. Similarly, the Ransomware, Uploading, SQL_injection, and XSS classes display AUC values between 0.96 and 0.99 across the runs. While these values are strong and suggest that the model generally discriminates well

between these attack types and others, the slight fluctuations indicate potential areas for further model refinement to achieve more consistent performance.

5- Challenges and Limitations

5.1. Computational Constraints

The integration of genetic algorithms (GA) to optimize neural network parameters, such as weights and biases, presents significant computational challenges. The following points highlight these constraints:

Genetic algorithms inherently require the evaluation of multiple candidate solutions across numerous generations to converge on an optimal or near-optimal solution. This process involves repeated training and evaluation of neural networks, which is computationally intensive. Each candidate solution represents a unique set of neural network parameters. The demand for computational resources such as CPUs, GPUs, and memory is significantly heightened. This can lead to long training times and the need for high-performance computing infrastructure.

The search space for neural network parameters is vast, especially when optimizing both weights and biases for multiple layers. In particular, having a moderately sized neural network with two hidden layers with 19 and 29 neurons results in more than a thousand parameters. The genetic algorithm must explore this high-dimensional space, which can lead to a combinatorial explosion in the number of evaluations needed to find optimal solutions.

The iterative nature of genetic algorithms, coupled with the need to evaluate a population of solutions in each generation, results in high time complexity. As the number of generations and population size increase, the time required to complete the optimization process grows substantially. This is particularly challenging when the neural network models are complex and the training datasets are large.

5.2. Dataset Size and Complexity

The foundational cornerstone of this research lies in the exploration of a substantial and intricate dataset. Comprising a vast array of 63 columns and an expansive 157,801 rows, the dataset encapsulates a rich diversity of data types and attributes. Each column serves as a unique dimension, providing a comprehensive perspective on the intricate landscape of cyber activities within the IIoT. The varied characteristics of data in cyber-physical systems require effective strategies to optimize performance across multiple dimensions. Additionally, the large volume of data demands careful consideration of computational efficiency and resource management. This highlights the importance of using optimization algorithms that can tackle these challenges while operating within the limits of available time and computational resources.

5.3. Algorithm Fine-Tuning

Optimizing algorithms like GA requires careful attention to the sensitivity of their tuning process. GA relies on several key parameters such as population size, mutation rate, crossover rate, and the number of generations, that affect their performance significantly. Fine-tuning these parameters is a critical step, as even small adjustments can have a major impact on the results. The challenge lies in finding the best balance, where the parameters work together effectively within the complexities of the dataset and the structure of the neural network.

For example, the population size and mutation rate play crucial roles in how well the GA can explore potential solutions and avoid getting stuck in suboptimal ones. Adjusting these parameters helps ensure that the algorithm thoroughly searches the solution space while maintaining the efficiency needed to find high-quality answers. Moreover, the components of the neural network architecture, such as the number of layers and neurons, add another layer of complexity and challenge to the tuning process. The GA needs to be carefully adapted to meet the specific needs of the neural network, requiring an in-depth understanding of both the GA and the neural network to achieve the best possible outcomes.

6- Future Work

Future research will focus on optimizing and fine-tuning neural network parameters to achieve higher accuracy in cyber-attack detection within the IIoT landscape. Efforts will explore advanced techniques for parameter selection, leveraging state-of-the-art methodologies to identify configurations that maximize detection efficacy.

Additionally, there will be a concerted effort to enhance training efficiency, aiming to accelerate convergence rates and reduce computational overhead. Novel approaches will be investigated to enable rapid deployment and adaptation of models in dynamic IIoT environments.

Furthermore, exploring new algorithms and optimization strategies holds promise for pushing current capabilities further. Integration of cutting-edge techniques such as meta-learning or adaptive learning rate methods could offer substantial improvements in both accuracy and efficiency, advancing robust cyber-attack detection systems.

7- References

[1] H. ElMaraghy, L. Monostori, G. Schuh, and W. ElMaraghy, “Evolution and future of manufacturing systems,” *CIRP Annals*, vol. 70, no. 2, pp. 635–658, Jan. 2021, doi: 10.1016/j.cirp.2021.05.008.

[2] M. Barrère, C. Hankin, N. Nicolaou, D. G. Eliades, and T. Parisini, “Measuring cyber-physical security in industrial control systems via minimum-effort attack strategies,” *Journal of Information Security and Applications*, vol. 52, Jun. 2020, doi: 10.1016/j.jisa.2020.102471.

[3] H. Kayan, M. Nunes, O. Rana, P. Burnap, and C. Perera, “Cybersecurity of Industrial Cyber-Physical Systems: A Review,” Jan. 2021, [Online]. Available: http://arxiv.org/abs/2101.03564

[4] A. M. Alnajim, S. Habib, M. Islam, S. M. Thwin, and F. Alotaibi, “A Comprehensive Survey of Cybersecurity Threats, Attacks, and Effective Countermeasures in Industrial Internet of Things,” Dec. 01, 2023, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/technologies11060161.

[5] M. N. Hoda, Bharati Vidyapeeth’s Institute of Computers Applications and Management Delhi, and Institute of Electrical and Electronics Engineers Delhi Section, *Proceedings of the 17th INDIACom; 2023 10th International Conference on Computing for Sustainable Global Development (15th-17th March, 2023) INDIACom-2023*.

[6] K. Stouffer, “Guide to Operational Technology (OT) Security,” 2023. doi: 10.6028/NIST.SP.800-82r3.

[7] T. S. Fun and A. Samsudin, “Recent technologies, security countermeasure and ongoing challenges of industrial internet of things (Iiot): A survey,” Oct. 01, 2021, *MDPI*. doi: 10.3390/s21196647.

[8] P. K. Malik *et al.*, “Industrial Internet of Things and its Applications in Industry 4.0: State of The Art,” *Comput Commun*, vol. 166, pp. 125–139, Jan. 2021, doi: 10.1016/j.comcom.2020.11.016.

[9] M. Javaid, A. Haleem, R. P. Singh, R. Suman, and E. S. Gonzalez, “Understanding the adoption of Industry 4.0 technologies in improving environmental sustainability,” *Sustainable Operations and Computers*, vol. 3, pp. 203–217, Jan. 2022, doi: 10.1016/j.susoc.2022.01.008.

[10] S. Mubarak *et al.*, “Industrial datasets with ICS testbed and attack detection using machine learning techniques,” *Intelligent Automation and Soft Computing*, vol. 31, no. 3, pp. 1345–1360, 2022, doi: 10.32604/IASC.2022.020801.

[11] J. G. Greener, S. M. Kandathil, L. Moffat, and D. T. Jones, “A guide to machine learning for biologists.”

[12] W. Jiang, “Machine Learning Methods to Detect Voltage Glitch Attacks on IoT/IIoT Infrastructures,” *Comput Intell Neurosci*, vol. 2022, 2022, doi: 10.1155/2022/6044071.

[13] V. Shah, “Revista Española de Documentación Científica Machine Learning Algorithms for Cybersecurity: Detecting and Preventing Threats”, doi: 10.5281/zenodo.10779509.

[14] P. Anand, Y. Singh, A. Selwal, M. Alazab, S. Tanwar, and N. Kumar, “IoT vulnerability assessment for sustainable computing: Threats, current solutions, and open challenges,” *IEEE Access*, vol. 8, pp. 168825–168853, 2020, doi: 10.1109/ACCESS.2020.3022842.

[15] I. A. Mohammed, “The Interaction Between Artificial Intelligence and Identity & Access Management: An Empirical study,” 2015. [Online]. Available: www.ijcrt.org

[16] N. Sun *et al.*, “Cyber Threat Intelligence Mining for Proactive Cybersecurity Defense: A Survey and New Perspectives,” *IEEE Communications Surveys and Tutorials*, vol. 25, no. 3, pp. 1748–1774, 2023, doi: 10.1109/COMST.2023.3273282.

[17] K. Shaukat *et al.*, “Performance comparison and current challenges of using machine learning techniques in cybersecurity,” May 01, 2020, *MDPI AG*. doi: 10.3390/en13102509.

[18] G. Apruzzese *et al.*, “The Role of Machine Learning in Cybersecurity,” *Digital Threats: Research and Practice*, vol. 4, no. 1, Mar. 2023, doi: 10.1145/3545574.

[19] J. Liu, K. Xiao, L. Luo, Y. Li, and L. Chen, “An intrusion detection system integrating network-level intrusion detection and host-level intrusion detection,” in *Proceedings - 2020 IEEE 20th International Conference on Software Quality, Reliability, and Security, QRS 2020*, Institute of Electrical and Electronics Engineers Inc., Dec. 2020, pp. 122–129. doi: 10.1109/QRS51102.2020.00028.

[20] M. Aljanabi, M. A. Ismail, and A. H. Ali, “Intrusion detection systems, issues, challenges, and needs,” *International Journal of Computational Intelligence Systems*, vol. 14, no. 1, pp. 560–571, 2021, doi: 10.2991/ijcis.d.210105.001.

[21] S. Kumar, S. Gupta, and S. Arora, “Research Trends in Network-Based Intrusion Detection Systems: A Review,” 2021, *Institute of Electrical and Electronics Engineers Inc.* doi: 10.1109/ACCESS.2021.3129775.

[22] D. Park, S. Kim, H. Kwon, D. Shin, and D. Shin, “Host-Based Intrusion Detection Model Using Siamese Network,” *IEEE Access*, vol. 9, pp. 76614–76623, 2021, doi: 10.1109/ACCESS.2021.3082160.

[23] Q. Liu, V. Hagenmeyer, and H. B. Keller, “A Review of Rule Learning-Based Intrusion Detection Systems and Their Prospects in Smart Grids,” 2021, *Institute of Electrical and Electronics Engineers Inc.* doi: 10.1109/ACCESS.2021.3071263.

[24] A. Thakkar and R. Lohiya, “A Review of the Advancement in Intrusion Detection Datasets,” in *Procedia Computer Science*, Elsevier B.V., 2020, pp. 636–645. doi: 10.1016/j.procs.2020.03.330.

[25] J. Díaz-Verdejo, J. Muñoz-Calle, A. E. Alonso, R. E. Alonso, and G. Madinabeitia, “On the Detection Capabilities of Signature-Based Intrusion Detection Systems in the Context of Web Attacks,” *Applied Sciences (Switzerland)*, vol. 12, no. 2, Jan. 2022, doi: 10.3390/app12020852.

[26] M. Antunes, L. Oliveira, A. Seguro, J. Veríssimo, R. Salgado, and T. Murteira, “Benchmarking Deep Learning Methods for Behaviour-Based Network Intrusion Detection,” *Informatics*, vol. 9, no. 1, Mar. 2022, doi: 10.3390/informatics9010029.

[27] H. Asad and I. Gashi, “Dynamical analysis of diversity in rule-based open source network intrusion detection systems,” *Empir Softw Eng*, vol. 27, no. 1, Jan. 2022, doi: 10.1007/s10664-021-10046-w.

[28] S. P. Thirimanne, L. Jayawardana, L. Yasakethu, P. Liyanaarachchi, and C. Hewage, “Deep Neural Network Based Real-Time Intrusion Detection System,” *SN Comput Sci*, vol. 3, no. 2, Mar. 2022, doi: 10.1007/s42979-022-01031-1.

[29] N. Sharma, A. Chakrabarti, and V. E. Balas, “Advances in Intelligent Systems and Computing 1042.” [Online]. Available: http://www.springer.com/series/11156

[30] S. Thapa and A. Mailewa, “EasyChair Preprint The Role of Intrusion Detection/Prevention Systems in Modern Computer Networks: A Review,” 2020.

[31] J. M. Kizza, *Guide to Computer Network Security*. in Texts in Computer Science. Cham: Springer International Publishing, 2024. doi: 10.1007/978-3-031-47549-8.

[32] S. A. Varghese, A. D. Ghadim, A. Balador, Z. Alimadadi, and P. Papadimitratos, “Digital Twin-based Intrusion Detection for Industrial Control Systems,” Jul. 2022, doi: 10.1109/PerComWorkshops53856.2022.9767492.

[33] M. Humayun, N. Z. Jhanjhi, M. N. Talib, M. H. Shah, and G. Sussendran, “Industry 4.0 and Cyber Security Issues and Challenges,” 2021.

[34] R. J. Raimundo and A. T. Rosário, “Cybersecurity in the Internet of Things in Industrial Management,” Feb. 01, 2022, *MDPI*. doi: 10.3390/app12031598.

[35] M. Culler, “Cyber Threat Landscape for Distribution Systems,” 2022.

[36] E. Izycki and E. W. Vianna, “Critical Infrastructure: A Battlefield for Cyber Warfare?”, doi: 10.34190/IWS.21.011.

[37] W. I. Khedr, A. E. Gouda, and E. R. Mohamed, “FMDADM: A Multi-Layer DDoS Attack Detection and Mitigation Framework Using Machine Learning for Stateful SDN-Based IoT Networks,” *IEEE Access*, 2023, doi: 10.1109/ACCESS.2023.3260256.

[38] Z. A. El Houda, B. Brik, A. Ksentini, and L. Khoukhi, “A MEC-Based Architecture to Secure IoT Applications using Federated Deep Learning,” *IEEE Internet of Things Magazine*, vol. 6, no. 1, pp. 60–63, Mar. 2023, doi: 10.1109/iotm.001.2100238.

[39] S. Kantimahanthi, J. V.D. Prasad, S. Chanamolu, and K. Kommaraju, “Machine Learning Approaches in Cyber Attack Detection and Characterization in IoT enabled Cyber-Physical Systems,” in *IDCIoT 2023 - International Conference on Intelligent Data Communication Technologies and Internet of Things, Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 136–142. doi: 10.1109/IDCIoT56793.2023.10053545.

[40] G. P. Bhandari, A. Lyth, A. Shalaginov, and T. M. Gronli, “Artificial Intelligence Enabled Middleware for Distributed Cyberattacks Detection in IoT-based Smart Environments,” in *Proceedings - 2022 IEEE International Conference on Big Data, Big Data 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 3023–3032. doi: 10.1109/BigData55660.2022.10020531.

[41] E. M. De Elias *et al.*, “A Hybrid CNN-LSTM Model for IIoT Edge Privacy-Aware Intrusion Detection,” in *2022 IEEE Latin-American Conference on Communications, LATINCOM 2022*, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/LATINCOM56090.2022.10000468.

[42] A. Khacha, R. Saadouni, Y. Harbi, and Z. Aliouat, “Hybrid Deep Learning-based Intrusion Detection System for Industrial Internet of Things,” in *ISIA 2022 - International Symposium on Informatics and its Applications, Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/ISIA55826.2022.9993487.

[43] O. Cheikhrouhou, O. Ben Fredj, N. Atitallah, and S. Hellal, “Intrusion Detection in Industrial IoT,” in *Proceedings of the 2022 15th IEEE International Conference on Security of Information and Networks, SIN 2022*, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/SIN56466.2022.9970535.

[44] Y. Otoum, V. Chamola, and A. Nayak, “Federated and Transfer Learning-Empowered Intrusion Detection for IoT Applications,” *IEEE Internet of Things Magazine*, vol. 5, no. 3, pp. 50–54, Nov. 2022, doi: 10.1109/iotm.001.2200048.

[45] P. Dini *et al.*, “Design and Testing Novel One-Class Classifier Based on Polynomial Interpolation with Application to Networking Security,” *IEEE Access*, vol. 10, pp. 67910–67924, 2022, doi: 10.1109/ACCESS.2022.3186026.

[46] A. Ghourabi, “A Security Model Based on LightGBM and Transformer to Protect Healthcare Systems From Cyberattacks,” *IEEE Access*, vol. 10, pp. 48890–48903, 2022, doi: 10.1109/ACCESS.2022.3172432.

[47] M. G. dos Santos, D. Ameyed, F. Petrillo, F. Jaafar, and M. Cheriet, “Internet of Things Architectures: A Comparative Study,” Apr. 2020, [Online]. Available: http://arxiv.org/abs/2004.12936

[48] D. Breitenbacher and Y. Elovici, “N-BaIoT-Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders.” [Online]. Available: http://archive.ics.uci.edu/ml/datasets/detec-

[49] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, “Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset,” Nov. 2018, [Online]. Available: http://arxiv.org/abs/1811.00701

[50] I. Vaccari, G. Chiola, M. Aiello, M. Mongelli, and E. Cambiaso, “Mqttset, a new dataset for machine learning techniques on mqtt,” *Sensors (Switzerland)*, vol. 20, no. 22, pp. 1–17, Nov. 2020, doi: 10.3390/s20226578.

[51] N. Moustafa, M. Keshk, E. Debie, and H. Janicke, “Federated TON IoT Windows Datasets for Evaluating AI-based Security Applications”, doi: 10.1109/TrustCom50675.2020.00114/20/$31.00.

[52] M. Al-Hawawreh, E. Sitnikova, and N. Aboutorab, “X-IIoTID: A Connectivity-Agnostic and Device-Agnostic Intrusion Data Set for Industrial Internet of Things,” *IEEE Internet Things J*, vol. 9, no. 5, pp. 3962–3977, Mar. 2022, doi: 10.1109/JIOT.2021.3102056.

[53] M. Zolanvari, M. A. Teixeira, L. Gupta, K. M. Khan, and R. Jain, “Machine Learning-Based Network Vulnerability Analysis of Industrial Internet of Things,” *IEEE Internet Things J*, vol. 6, no. 4, pp. 6822–6834, Aug. 2019, doi: 10.1109/JIOT.2019.2912022.

[54] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, “Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning,” *IEEE Access*, vol. 10, pp. 40281–40306, 2022, doi: 10.1109/ACCESS.2022.3165809.

[55] A. B. de Neira, B. Kantarci, and M. Nogueira, “Distributed denial of service attack prediction: Challenges, open issues and opportunities,” Feb. 01, 2023, *Elsevier B.V.* doi: 10.1016/j.comnet.2022.109553.

[56] V. Nagaraju, A. Raaza, V. Rajendran, and D. Ravikumar, “Deep learning binary fruit fly algorithm for identifying SYN flood attack from TCP/IP,” *Mater Today Proc*, vol. 80, pp. 3086–3091, Jan. 2023, doi: 10.1016/j.matpr.2021.07.171.

[57] Y. Wang, J. Ding, T. Zhang, Y. Xiao, and X. Hei, “From Replay to Regeneration: Recovery of UDP Flood Network Attack Scenario Based on SDN,” *Mathematics*, vol. 11, no. 8, Apr. 2023, doi: 10.3390/math11081897.

[58] P. Razumov, K. Lyashenko, L. Cherckesova, E. Revyakina, I. Yengibaryan, and A. Revyakin, “Development of a system for protecting against DDoS attacks at the L7 level of the OSI model - HTTP Flood,” in *E3S Web of Conferences*, EDP Sciences, Jul. 2023. doi: 10.1051/e3sconf/202340203008.

[59] R. Anusuya, C. Prathima, M. Ramkumar Prabhu, and J. R. Arun Kumar, “Detection of TCP, UDP and ICMP DDOS attacks in SDN Using Machine Learning approach,” 2023.

[60] X. Etxezarreta, I. Garitano, M. Iturbe, and U. Zurutuza, “Low delay network attributes randomization to proactively mitigate reconnaissance attacks in industrial control systems,” *Wireless Networks*, 2023, doi: 10.1007/s11276-022-03212-5.

[61] J. M. Pittman, “Machine Learning and Port Scans: A Systematic Review,” Jan. 2023, [Online]. Available: http://arxiv.org/abs/2301.13581

[62] M. Laštovička, M. Husák, P. Velan, T. Jirsík, and P. Čeleda, “Passive operating system fingerprinting revisited: Evaluation and current challenges,” *Computer Networks*, vol. 229, Jun. 2023, doi: 10.1016/j.comnet.2023.109782.

[63] C. Kalaani, “OWASP ZAP vs Snort for SQLi Vulnerability Scanning,” 2023. [Online]. Available: https://digitalcommons.georgiasouthern.edu/etd

[64] U. O. Obonna *et al.*, “Detection of Man-in-the-Middle (MitM) Cyber-Attacks in Oil and Gas Process Control Networks Using Machine Learning Algorithms,” *Future Internet*, vol. 15, no. 8, Aug. 2023, doi: 10.3390/fi15080280.

[65] E. Blancaflor, R. Ambayon, E. S. Gran, D. D. Medallo, and R. J. Ramirez, “Exploitation Simulation of DNS Spoofing and Ransomware In a Virtualized Android Device,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Mar. 2023, pp. 213–219. doi: 10.1145/3592307.3592341.

[66] F. Mvah, V. Kengne Tchendji, C. Tayou Djamegni, A. H. Anwar, D. K. Tosh, and C. Kamhoua, “GaTeBaSep: game theory-based security protocol against ARP spoofing attacks in software-defined networks,” *Int J Inf Secur*, 2023, doi: 10.1007/s10207-023-00749-0.

[67] A. A. Habib, M. K. Hasan, A. Alkhayyat, S. Islam, R. Sharma, and L. M. Alkwai, “False data injection attack in smart grid cyber physical system: Issues, challenges, and future direction,” *Computers and Electrical Engineering*, vol. 107, Apr. 2023, doi: 10.1016/j.compeleceng.2023.108638.

[68] J. Kaur, U. Garg, and G. Bathla, “Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review,” *Artif Intell Rev*, vol. 56, no. 11, pp. 12725–12769, Nov. 2023, doi: 10.1007/s10462-023-10433-3.

[69] R. Pedro, D. Castro, P. Carreira, and N. Santos, “From Prompt Injections to SQL Injection Attacks: How Protected is Your LLM-Integrated Web Application?,” Aug. 2023, [Online]. Available: http://arxiv.org/abs/2308.01990

[70] R. Al-Khannak and S. S. Nehal, “Penetration Testing for the Cloud-Based Web Application,” *WSEAS TRANSACTIONS ON COMPUTERS*, vol. 22, pp. 104–113, Aug. 2023, doi: 10.37394/23205.2023.22.13.

[71] A. Amira, A. Derhab, E. B. Karbab, and O. Nouali, “A Survey of Malware Analysis Using Community Detection Algorithms,” *ACM Comput Surv*, vol. 56, no. 2, Sep. 2023, doi: 10.1145/3610223.

[72] N. Kandpal, M. Jagielski, F. Tramèr, and N. Carlini, “Backdoor Attacks for In-Context Learning with Language Models,” Jul. 2023, [Online]. Available: http://arxiv.org/abs/2307.14692

[73] I. Alkhwaja *et al.*, “Password Cracking with Brute Force Algorithm and Dictionary Attack Using Parallel Programming,” *Applied Sciences (Switzerland)*, vol. 13, no. 10, May 2023, doi: 10.3390/app13105979.

[74] C. Dameff *et al.*, “Ransomware Attack Associated With Disruptions at Adjacent Emergency Departments in the US,” *JAMA Netw Open*, vol. 6, no. 5, p. e2312270, May 2023, doi: 10.1001/jamanetworkopen.2023.12270.

[75] A. Subasi, *Practical Machine Learning for Data Analysis Using Python*. Elsevier, 2020. doi: 10.1016/B978-0-12-821379-7.00008-4.

[76] D. Alexander TEDJOPUMOMO *et al.*, “A survey on modern deep neural network for traffic prediction: A survey on modern deep neural network for traffic prediction: Trends, methods and challenges Trends, methods and challenges Citation Citation A survey on modern deep neural network for traffic prediction: Trends, methods and challenges A Survey on Modern Deep Neural Network for Traffic Prediction: Trends, Methods and Challenges,” 2022. [Online]. Available: https://ink.library.smu.edu.sg/sis_research/5995

[77] S. Katoch, S. S. Chauhan, and V. Kumar, “A review on genetic algorithm: past, present, and future,” *Multimed Tools Appl*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021, doi: 10.1007/s11042-020-10139-6.

[78] M. Alam, M. D. Samad, L. Vidyaratne, A. Glandon, and K. M. Iftekharuddin, “Survey on Deep Neural Networks in Speech and Vision Systems,” *Neurocomputing*, vol. 417, pp. 302–321, Dec. 2020, doi: 10.1016/j.neucom.2020.07.053.

[79] S. M. Kasongo, “A deep learning technique for intrusion detection system using a Recurrent Neural Networks based framework,” *Comput Commun*, vol. 199, pp. 113–125, Feb. 2023, doi: 10.1016/j.comcom.2022.12.010.

[80] X. Xiang and S. Foo, “Recent Advances in Deep Reinforcement Learning Applications for Solving Partially Observable Markov Decision Processes (POMDP) Problems: Part 1—Fundamentals and Applications in Games, Robotics and Natural Language Processing,” Sep. 01, 2021, *MDPI*. doi: 10.3390/make3030029.

[81] S. Dalal *et al.*, “Extremely boosted neural network for more accurate multi-stage Cyber attack prediction in cloud computing environment,” *Journal of Cloud Computing*, vol. 12, no. 1, Dec. 2023, doi: 10.1186/s13677-022-00356-9.

[82] M. Aljabri *et al.*, “Intelligent techniques for detecting network attacks: Review and research directions,” Nov. 01, 2021, *MDPI*. doi: 10.3390/s21217070.

[83] Mitsuo Gen and Lin Lin, “Genetic Algorithms and Their Applications,” 2023.

[84] M. Mosayebi and M. Sodhi, “Tuning genetic algorithm parameters using design of experiments,” in *GECCO 2020 Companion - Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, Association for Computing Machinery, Inc, Jul. 2020, pp. 1937–1944. doi: 10.1145/3377929.3398136.

[85] M. Greenacre *et al.*, “Economics Working Paper Series Principal component analysis Principal Component Analysis,” 2023.

[86] F. L. Gewers *et al.*, “Principal Component Analysis: A Natural Approach to Data Exploration,” Apr. 2018, doi: 10.1145/3447755.

[87] R. Zebari, A. Abdulazeez, D. Zeebaree, D. Zebari, and J. Saeed, “A Comprehensive Review of Dimensionality Reduction Techniques for Feature Selection and Feature Extraction,” *Journal of Applied Science and Technology Trends*, vol. 1, no. 1, pp. 56–70, May 2020, doi: 10.38094/jastt1224.

[88] C. Giannelli and M. Picone, “Editorial ‘Industrial IoT as IT and OT Convergence: Challenges and Opportunities,’” Mar. 01, 2022, *MDPI*. doi: 10.3390/iot3010014.

[89] F. E. Botchey, Z. Qin, K. Hughes-Lartey, and K. E. Ampomah, “Predicting Fraud in Mobile Money Transactions using Machine Learning: The Effects of Sampling Techniques on the Imbalanced Dataset,” *Informatica (Slovenia)*, vol. 45, no. 7, pp. 45–56, 2021, doi: 10.31449/inf.v45i7.3179.

[90] M. S. Mahmud, J. Z. Huang, S. Salloum, T. Z. Emara, and K. Sadatdiynov, “A survey of data partitioning and sampling methods to support big data analysis,” Jun. 01, 2020, *Tsinghua University Press*. doi: 10.26599/BDMA.2019.9020015.

[91] G. T. Reddy *et al.*, “Analysis of Dimensionality Reduction Techniques on Big Data,” *IEEE Access*, vol. 8, pp. 54776–54788, 2020, doi: 10.1109/ACCESS.2020.2980942.

[92] W. Jia, M. Sun, J. Lian, and S. Hou, “Feature dimensionality reduction: a review,” *Complex and Intelligent Systems*, vol. 8, no. 3, pp. 2663–2693, Jun. 2022, doi: 10.1007/s40747-021-00637-x.

[93] B. M. S. Hasan and A. M. Abdulazeez, “A Review of Principal Component Analysis Algorithm for Dimensionality Reduction,” *Journal of Soft Computing and Data Mining*, vol. 2, no. 1, pp. 20–30, Apr. 2021, doi: 10.30880/jscdm.2021.02.01.003.

[94] J. Bharadiya and J. P. Bharadiya, “A Tutorial on Principal Component Analysis for Dimensionality Reduction in Machine Learning,” *Article in International Journal of Innovative Research in Science Engineering and Technology*, vol. 8, no. 5, 2023, doi: 10.5281/zenodo.8002436.

[95] V. S. Konduri, T. J. Vandal, S. Ganguly, and A. R. Ganguly, “Data Science for Weather Impacts on Crop Yield,” *Front Sustain Food Syst*, vol. 4, May 2020, doi: 10.3389/fsufs.2020.00052.

[96] baeldung and Michal Aibin, “How Many Principal Components to Take in PCA?,” https://www.baeldung.com/cs/pca.

[97] C. Richter, K. Mcguinness, L. Gualano, N. E. O’connor, and K. Moran, “Identification of an optimal principal components analysis threshold to describe jump height accurately using vertical ground reaction forces.”

[98] M. G. M. Abdolrasol *et al.*, “Artificial neural networks based optimization techniques: A review,” Nov. 01, 2021, *MDPI*. doi: 10.3390/electronics10212689.

[99] C. Ciancio, G. Ambrogio, F. Gagliardi, and R. Musmanno, “Heuristic techniques to optimize neural network architecture in manufacturing applications,” *Neural Comput Appl*, vol. 27, no. 7, pp. 2001–2015, Oct. 2016, doi: 10.1007/s00521-015-1994-9.

[100] M. Abd Elaziz *et al.*, “Advanced metaheuristic optimization techniques in applications of deep neural networks: a review,” Nov. 01, 2021, *Springer Science and Business Media Deutschland GmbH*. doi: 10.1007/s00521-021-05960-5.

[101] P. Ferber, M. Helmert, and J. Hoffmann, “Neural network heuristics for classical planning: A study of hyperparameter space,” in *Frontiers in Artificial Intelligence and Applications*, IOS Press BV, Aug. 2020, pp. 2346–2353. doi: 10.3233/FAIA200364.

[102] Y. Zhou, T. Pang, K. Liu, C. H. Martin, M. W. Mahoney, and Y. Yang, “Temperature Balancing, Layer-wise Weight Analysis, and Neural Network Training.”

[103] D. Singh Kalra and M. Barkeshli, “Phase diagram of early training dynamics in deep networks: effect of the learning rate, depth, and width.”

[104] X. Yuan, P. Savarese, and M. Maire, “Accelerated Training via Incrementally Growing Neural Networks using Variance Transfer and Learning Rate Adaptation.”

[105] Olympia Roeva, Stefka Fidanova, and Marcin Paprzycki, *Influence of the Population Size on the Genetic Algorithm Performance in Case of Cultivation Process Modelling*.

[106] D. Valero-Carreras, J. Alcaraz, and M. Landete, “Comparing two SVM models through different metrics based on the confusion matrix,” *Comput Oper Res*, vol. 152, Apr. 2023, doi: 10.1016/j.cor.2022.106131.

[107] D. Fourure, M. U. Javaid, N. Posocco, and S. Tihon, “Anomaly Detection: How to Artificially Increase your F1-Score with a Biased Evaluation Protocol,” Jun. 2021, [Online]. Available: http://arxiv.org/abs/2106.16020

[108] M. Heydarian, T. E. Doyle, and R. Samavi, “MLCM: Multi-Label Confusion Matrix,” *IEEE Access*, vol. 10, pp. 19083–19095, 2022, doi: 10.1109/ACCESS.2022.3151048.

[109] F. S. Nahm, “Receiver operating characteristic curve: overview and practical use for clinicians,” *Korean J Anesthesiol*, vol. 75, no. 1, pp. 25–36, Feb. 2022, doi: 10.4097/kja.21209.

[110] P. Qiu, Z. Xia, and L. You, “Process Monitoring ROC Curve for Evaluating Dynamic Screening Methods,” *Technometrics*, vol. 62, no. 2, pp. 236–248, Apr. 2020, doi: 10.1080/00401706.2019.1604434.

8- Appendices

Appendix A: PCA Detailed Results

Table 16: PCA Detailed Results Part 1

Original Features	PCA 1	PCA 2	PCA 3	PCA 4	PCA 5	PCA 6	PCA 7
frame.time	0.09349	-0.601528	0.455655	-0.001028	0.283592	-0.14476	0.205268
ip.src_host	-0.288125	-0.054439	0.631606	-0.086858	-0.090714	0.159032	-0.392949
ip.dst_host	-0.082474	0.024128	0.070564	-0.03284	-0.023545	-0.05489	0.678382
arp.dst.proto_ipv4	-0.001874	0.001061	-0.000654	0.000566	-0.000381	-0.001127	0.002499
arp.opcode	-0.000009	0.000009	-0.000007	0.000003	-0.000007	-0.000002	0.000005
arp.hw.size	-0.000022	0.00002	-0.000016	0.000006	-0.000015	-0.000005	0.000014
arp.src.proto_ipv4	-0.000004	0.000006	-0.000003	0	-0.000004	0	0.000003
icmp.checksum	-0.113223	-0.031427	0.256162	-0.034927	0.027714	0.048614	0.055483
icmp.seq_le	-0.13159	-0.025707	0.277205	-0.031764	0.009458	0.060086	0.094638
icmp.transmit_timestamp	-0.000004	0.000005	-0.000005	0.000001	-0.000005	0.000001	-0.000002
icmp.unused	0	0	0	0	0	0	0
http.file_data	0.104203	0.0336	0.060824	0.220581	-0.01252	0.120201	0.023399
http.content_length	0.043852	0.014057	0.026088	0.092417	-0.005064	0.048026	0.010308
http.request.uri.query	0.023285	0.004299	0.008177	0.059065	-0.009384	0.052921	-0.01636

http.request.method	0.01507	0.003428	0.005103	0.032774	0.00079	0.047502	-0.000347
http.referer	0.004761	0.000251	0.002669	0.007421	0.004095	0.016301	0.003933
http.request.full_uri	0.068738	0.010394	0.027384	0.152034	-0.007108	0.165348	-0.020031
http.request.version	0.013011	0.003785	0.002441	0.030275	-0.003162	0.043865	-0.004453
http.response	0.000004	0.000003	0.000003	0.000008	-0.000001	0.000003	0.000004
http.tls_port	0	0	0	0	0	0	0
tcp.ack	0.213616	0.064719	0.109048	0.065568	-0.135692	-0.689572	-0.336201
tcp.ack_raw	0.59776	-0.071656	0.111988	-0.321375	-0.021057	-0.144321	0.109771
tcp.checksum	0.293137	0.011654	-0.005474	-0.236578	0.081432	0.066374	-0.122318
tcp.connection.fin	0.000002	0	-0.000001	-0.000004	0.000002	0	-0.000001
tcp.connection.rst	-0.000002	0.000003	-0.000004	-0.000013	-0.000003	-0.000001	0.000016
tcp.connection.syn	-0.000002	0.000003	-0.000005	-0.000004	-0.000004	0	-0.000026
tcp.connection.synack	0.000001	0.000001	-0.000001	-0.000002	0.000006	0.000003	0
tcp.dstport	0.172817	-0.047558	-0.042703	-0.386209	0.008966	0.075748	0.172139
tcp.flags	0.008303	-0.002146	-0.00729	-0.016343	0.00206	0.009481	-0.003584
tcp.flags.ack	0.000025	0.000016	-0.000007	-0.000009	0.000002	0.000004	0.000016

tcp.len	0.094585	0.007401	0.05058	0.127448	-0.145505	0.039899	0.05854
tcp.options	0.330313	0.173074	0.060394	0.339346	0.717661	0.241244	-0.113692
tcp.payload	0.344172	-0.020949	0.176447	0.395522	-0.512367	0.326837	0.106963
tcp.seq	0.222637	0.031478	0.122895	0.317885	-0.190451	-0.195169	0.004575
tcp.srcport	0.105532	-0.638266	-0.32863	-0.092994	-0.125443	0.290122	-0.314327
udp.port	-0.00025	-0.000091	0.000027	0.000274	0.000183	-0.000496	0.000816
udp.stream	-0.182301	-0.422287	-0.230841	0.435289	0.094477	-0.281133	0.112511
udp.time_delta	-0.000107	-0.000049	0.000041	0.000128	0.000058	-0.000188	0.000256
dns.qry.name	-0.035222	0.015753	0.024667	0.01062	-0.038972	0.011113	0.100545
dns.qry.name.len	-0.000463	-0.000162	0.00006	0.000491	0.000317	-0.000892	0.001578
dns.qry.qu	-0.000002	0.000003	-0.000003	0.000001	-0.000003	0.000001	-0.000001
dns.qry.type	0	0	0	0	0	0	0
dns.retransmission	-0.000052	-0.000019	0.000007	0.000058	0.000034	-0.000101	0.000187
dns.retransmit_request	0	0	0	0	0	0	0
dns.retransmit_request_in	0	0	0	0	0	0	0
mqtt.conack.flags	0.018089	-0.022203	0.000869	-0.020945	-0.051353	0.056426	0.022254

mqtt.conflag.cleansess	0.000001	-0.000001	0	-0.000001	-0.000002	0.000002	0.000002
mqtt.conflags	0.000025	-0.000025	0.000005	-0.00003	-0.00007	0.00007	0.000054
mqtt.hdrflags	0.008721	-0.008515	0.001879	-0.01024	-0.028157	0.024738	0.019568
mqtt.len	0.00755	-0.007255	0.001718	-0.009033	-0.024955	0.021957	0.017999
mqtt.msg_decoded_as	0	0	0	0	0	0	0
mqtt.msg	0.02363	-0.022686	0.005413	-0.028198	-0.078087	0.068688	0.056356
mqtt.msgtype	0.022038	-0.021158	0.004874	-0.024225	-0.068421	0.055407	0.045045
mqtt.proto_len	0.002182	-0.002159	0.000429	-0.002619	-0.006031	0.006016	0.004672
mqtt.protoname	0.01801	-0.017823	0.003538	-0.021619	-0.049783	0.049662	0.038566
mqtt.topic	0.022327	-0.021433	0.005094	-0.026716	-0.073829	0.06491	0.053311
mqtt.topic_len	0.000139	-0.000133	0.000032	-0.000166	-0.000458	0.000403	0.000331
mqtt.ver	0.002182	-0.002159	0.000429	-0.002619	-0.006031	0.006016	0.004672
mbtcp.len	0	0	0	0	0	0	0
mbtcp.trans_id	0	0	0	0	0	0	0
mbtcp.unit_id	0	0	0	0	0	0	0

Table 17: PCA Detailed Results Part 2

Original Features	**PCA 8**	**PCA 9**	**PCA 10**	**PCA 11**	**PCA 12**	**PCA 13**	**PCA 14**

frame.time	-0.109273	0.229408	0.123213	-0.240094	-0.023835	0.073326	-0.053409
ip.src_host	0.076235	-0.184497	-0.011344	0.364831	0.061332	-0.138153	0.018703
ip.dst_host	0.400821	-0.34236	-0.264281	-0.049395	-0.116473	0.013169	-0.081112
arp.dst.proto_ipv4	-0.005071	0.006483	0.001857	-0.008145	0.001102	0.005309	-0.001566
arp.opcode	-0.00002	0.00002	0.000003	-0.000024	0.000007	0.000019	-0.000002
arp.hw.size	-0.000047	0.000051	0.000009	-0.000061	0.000015	0.000047	-0.000006
arp.src.proto_ipv4	-0.000012	0.000013	0.000001	-0.000017	0.000003	0.000012	-0.000002
icmp.checksum	0.008603	0.054267	-0.045558	-0.272831	-0.066584	0.105469	-0.039509
icmp.seq_le	0.1188	-0.101931	-0.113177	-0.034367	-0.062691	-0.044798	0.136446
icmp.transmit_timestamp	-0.000005	0.000002	-0.000002	-0.000002	0.000003	0.000004	0.000002
icmp.unused	0	0	0	0	0	0	0
http.file_data	0.00495	-0.280372	0.264995	-0.275423	0.288071	-0.185544	0.172085
http.content_length	0.00158	-0.116725	0.112409	-0.114126	0.117077	-0.08057	0.068622
http.request.uri.query	0.023404	-0.107005	0.034082	-0.132078	0.158583	-0.011031	0.181516
http.request.method	0.006406	-0.051755	0.022752	-0.045184	0.088927	0.051203	0.045132

http.referer	-0.001457	-0.010272	0.006055	-0.000464	0.024928	0.036019	-0.005095
http.request.full_uri	0.045153	-0.236556	0.083239	-0.244879	0.360183	0.104704	0.278258
http.request.version	0.009066	-0.048993	0.019052	-0.047565	0.075264	0.03485	0.046818
http.response	-0.000003	-0.000007	0.000016	-0.000003	0	-0.000012	-0.00001
http.tls_port	0	0	0	0	0	0	0
tcp.ack	0.098291	-0.397597	0.079901	-0.211068	-0.296719	0.048602	-0.09157
tcp.ack_raw	-0.219153	-0.13584	-0.204825	0.349052	0.392331	0.21774	0.069759
tcp.checksum	0.777338	0.283389	0.369753	0.054505	0.049362	0.040698	0.005676
tcp.connection.fin	-0.000002	0.000004	-0.000007	0.000004	0.000001	-0.000004	0.000007
tcp.connection.rst	0.000003	0.000011	0.000011	-0.000007	0.000015	-0.000011	0.000006
tcp.connection.syn	0.000015	-0.000013	-0.000001	-0.000013	-0.000026	0.000009	-0.000024
tcp.connection.synack	-0.000002	0	-0.000002	0.000005	0.000001	0.000001	0
tcp.dstport	-0.230159	-0.138883	0.338588	0.004613	-0.23012	-0.681746	0.031493
tcp.flags	0.002173	0.023895	-0.015039	-0.015478	-0.014479	-0.033947	0.015258
tcp.flags.ack	-0.000021	0.000008	-0.000007	0.000021	0.000028	-0.000003	0.000022
tcp.len	-0.026685	-0.005948	0.125702	-0.043789	-0.049018	-0.024778	-0.003766

tcp.options	-0.006796	-0.157035	-0.137815	0.122953	-0.28625	-0.035903	-0.06644
tcp.payload	-0.040479	0.026679	0.147335	0.092879	-0.175037	0.086282	-0.352517
tcp.seq	0.109158	0.438959	-0.421555	-0.008274	0.003534	-0.452967	0.347136
tcp.srcport	0.152748	-0.228311	-0.364096	-0.151222	-0.138899	-0.059618	0.006058
udp.port	-0.000442	0.000691	0.000674	-0.00005	0.000346	0.000297	-0.00005
udp.stream	0.057621	-0.066082	0.301447	0.458536	0.129013	-0.06168	0.034845
udp.time_delta	-0.000053	0.000071	0.000271	0.000453	0.000207	-0.000096	0.000076
dns.qry.name	0.191771	-0.257797	-0.108752	0.326803	0.003793	-0.150704	0.062791
dns.qry.name.len	-0.000633	0.001009	0.001092	0.0003	0.000622	0.000285	0.000057
dns.qry.qu	-0.000003	0.000001	-0.000001	-0.000001	0.000002	0.000002	0.000001
dns.qry.type	0	0	0	0	0	0	0
dns.retransmission	-0.00004	0.000068	0.000109	0.000114	0.000074	-0.000011	0.000031
dns.retransmit_request	0	0	0	0	0	0	0
dns.retransmit_request_in	0	0	0	0	0	0	0
mqtt.conack.flags	-0.021611	0.001259	0.002128	0.022537	0.015765	0.046684	-0.26659
mqtt.conflag.cleansess	-0.000002	0	0.000002	0.000001	0	-0.000001	-0.000013

mqtt.conflags	-0.00005	0.000009	0.000061	0.00004	0.000008	-0.000018	-0.000356
mqtt.hdrflags	-0.021041	-0.001797	0.038833	0.031782	-0.101564	0.076454	0.12008
mqtt.len	-0.019568	-0.00244	0.038392	0.030326	-0.100925	0.076053	0.129955
mqtt.msg_decoded_as	0	0	0	0	0	0	0
mqtt.msg	-0.061177	-0.007619	0.120421	0.095074	-0.316696	0.238572	0.40862
mqtt.msgtype	-0.047563	0.001861	0.079412	0.069748	-0.220558	0.146162	0.259025
mqtt.proto_len	-0.004333	0.000792	0.005238	0.003458	0.000658	-0.00158	-0.030702
mqtt.protoname	-0.035768	0.006541	0.043233	0.028545	0.005427	-0.013046	-0.253433
mqtt.topic	-0.057968	-0.007247	0.113908	0.089889	-0.29956	0.225492	0.386809
mqtt.topic_len	-0.00036	-0.000045	0.000707	0.000558	-0.00186	0.0014	0.002402
mqtt.ver	-0.004333	0.000792	0.005238	0.003458	0.000658	-0.00158	-0.030702
mbtcp.len	0	0	0	0	0	0	0
mbtcp.trans_id	0	0	0	0	0	0	0
mbtcp.unit_id	0	0	0	0	0	0	0