MULTISTAGE INTRUSION DETECTION FOR ADVERSARIAL ATTACKS BASED ON RDLS MODEL IN IOT

A PROJECT REPORT

Submitted by

RHUTHVIKA DASI: 2020103557

SINDHURA M: 2020103569

SUNDARESAN HAARINI AMRUTDHA:2020103574

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY CAMPUS

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2024

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**MULTISTAGE INTRUSION DETECTION SYSTEM FOR ADVERSARIAL ATTACKS BASED ON RDLS MODEL IN IOT"** is the bonafide work of "**Rhuthvika Dasi (2020103557), Sindhura M (2020103569) and Sundaresan Haarini Amrutdha (2020103557)**" who carried out the project work under my supervision, for the fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

Place : Chennai

Date : 25/04/2024

SIGNATURE

Dr.S.Bose

SUPERVISOR

Professor

Department of Computer Science and Engineering

Anna University, Chennai – 600 025.

COUNTER SIGNED

Dr. S. Valli

HEAD OF THE DEPARTMENT

Computer Science and Engineering,

Anna University, Chennai - 600 025.

ACKNOWLEDGEMENT

We express our deep gratitude to our guide, **Dr.S.Bose, Professor**, Department of Computer Science and Engineering, for guiding us through every phase of the project. We appreciate his thoroughness, tolerance and ability to share his knowledge with us. We would also like to thank him for his kind support and for providing necessary facilities to carry out the work.

We are extremely grateful to **Dr. S. Valli, Professor & Head of the Department**, Department of Computer Science and Engineering, Anna University, Chennai – 25, for extending the facilities of the Department towards our project and for her unstinting support.

Our express our thanks to the panel of reviewers **Dr.S.Chitrakala, Professor**, Department of Computer Science and Engineering, **Dr.R.Baskaran, Professor**, Department of Computer Science and Engineering for their valuable suggestions and critical reviews throughout the course of our project.

We express our thanks to all other teaching and non-teaching staff who helped us in one way or other for the successful completion of the project. We would also like to thank our parents, family and friends for their indirect contribution in the successful completion of this project.

Rhuthvika Dasi **Sindhura M** **Sundaresan Haarini Amrutdha**

ABSTRACT- ENGLISH

The widespread adoption of Internet of Things (IoT) technologies has transformed smart city applications, offering new possibilities for efficiency and convenience. However, the use of deep learning (DL) models for detecting cyber-attacks in these IoT environments presents significant security challenges. Adversarial attacks pose a unavoidable threat, exploiting weaknesses in DL models to deceive them into making incorrect predictions. This paper explores how adversarial training techniques can make DL models stronger against such attacks in IoT-based smart city contents. Employing a hybrid ensemble model that combines DL algorithms like Deep Neural Networks (DNN) with conventional machine learning methods such as Random Forest, Support Vector Machine (SVM), and Logistic Regression, initial model training is conducted. Subsequently, various adversarial attack strategies, including Fast Gradient Sign Method (FGSM), Basic Iterative Method (BIM), Jacobian-based Saliency Map Attack (JSMA), Projected Gradient Descent (PGD), and Deep Fool (DF), are employed to generate adversarial test data. This newly generated test data is utilized to assess the robustness of the trained model against adversarial attacks. Moreover, the model is re-trained using an expanded training dataset containing both original and adversarial test data to strengthen its ability to resist attacks. Through the implementation of adversarial training techniques, this study aims to mitigate the risks posed by sophisticated adversaries seeking to manipulate data and breach security measures within smart city infrastructures. Experimental results reveal an initial model accuracy of 99%, based on the DS2OS dataset. Following retraining with an expanded dataset containing both original and adversarial test data, the model showed 98% accuracy. These findings contribute to advancing the development of more secure DL-based adversarial attack detection systems for IoT-enabled smart cities.

ABSTRACT – TAMIL

இன்டர்நெட் ஆஃப் திங்ஸ் (IoT) தொழில்நுட்பங்களின் பரவலான தத்தெடுப்பு ஸ்மார்ட் சிட்டி பயன்பாடுகளை மாற்றியமைத்து, செயல்திறன் மற்றும் வசதிக்கான புதிய சாத்தியங்களை வழங்குகிறது. இருப்பினும், இந்த IoT சூழல்களில் சைபர் தாக்குதல்களைக் கண்டறிவதற்கான ஆழமான கற்றல் (DL) மாதிரிகளைப் பயன்படுத்துவது குறிப்பிடத்தக்க பாதுகாப்பு சவால்களை முன்வைக்கிறது. எதிரிகளின் தாக்குதல்கள் தவிர்க்க முடியாத அச்சுறுத்தலை ஏற்படுத்துகின்றன, DL மாடல்களில் உள்ள பலவீனங்களைப் பயன்படுத்தி தவறான கணிப்புகளைச் செய்து அவர்களை ஏமாற்றுகின்றன. IoT-அடிப்படையிலான ஸ்மார்ட் சிட்டி உள்ளடக்கங்களில் இத்தகைய தாக்குதல்களுக்கு எதிராக டிஎல் மாடல்களை எவ்வாறு எதிரிடையான பயிற்சி நுட்பங்கள் வலிமையாக்க முடியும் என்பதை இந்தக் கட்டுரை ஆராய்கிறது. ரேண்டம் ஃபாரஸ்ட், சப்போர்ட் வெக்டர் மெஷின் (எஸ்விஎம்) மற்றும் லாஜிஸ்டிக் ரிக்ரஷன் போன்ற வழக்கமான இயந்திர கற்றல் முறைகளுடன் டீப் நியூரல் நெட்வொர்க்குகள் (டிஎன்என்) போன்ற டிஎல் அல்காரிதம்களை இணைக்கும் ஹைப்ரிட் குழும மாதிரியைப் பயன்படுத்துதல், ஆரம்ப மாதிரி பயிற்சி நடத்தப்படுகிறது. அதைத் தொடர்ந்து, ஃபாஸ்ட் கிரேடியன்ட் சைன் முறை (FGSM), அடிப்படை தொடர்பு முறை (BIM), ஜேகோபியன் அடிப்படையிலான Saliency Map Attack (JSMA), ப்ராஜெக்ட் க்ரேடியன்ட் டிசென்ட் (PGD) மற்றும் டீப் ஃபூல் (DF) உள்ளிட்ட பல்வேறு எதிர் தாக்குதல் உத்திகள் பயன்படுத்தப்படுகின்றன. எதிர்மறையான சோதனைத் தரவை உருவாக்குகிறது. புதிதாக உருவாக்கப்பட்ட இந்த சோதனைத் தரவு எதிரி தாக்குதல்களுக்கு எதிராக பயிற்சியளிக்கப்பட்ட மாதிரியின் வலிமையை மதிப்பிடுவதற்குப் பயன்படுத்தப்படுகிறது. மேலும், தாக்குதல்களை எதிர்க்கும் திறனை வலுப்படுத்த, அசல் மற்றும் எதிரியான சோதனைத் தரவைக் கொண்ட விரிவாக்கப்பட்ட பயிற்சி தரவுத்தொகுப்பைப் பயன்படுத்தி மாதிரி மீண்டும் பயிற்சியளிக்கப்படுகிறது. விரோதப் பயிற்சி நுட்பங்களைச் செயல்படுத்துவதன் மூலம், ஸ்மார்ட் சிட்டி உள்கட்டமைப்புகளுக்குள் தரவைக் கையாளவும் பாதுகாப்பு நடவடிக்கைகளை மீறவும் முயற்சிக்கும் அதிநவீன எதிரிகளால் ஏற்படும் அபாயங்களைக் குறைப்பதை இந்த ஆய்வு நோக்கமாகக் கொண்டுள்ளது. DS2OS தரவுத்தொகுப்பின் அடிப்படையில் 99% ஆரம்ப மாதிரி துல்லியத்தை சோதனை முடிவுகள் வெளிப்படுத்துகின்றன. அசல் மற்றும் எதிர்மறையான சோதனைத் தரவு இரண்டையும் உள்ளடக்கிய விரிவாக்கப்பட்ட தரவுத்தொகுப்புடன் மீண்டும் பயிற்சியளித்ததைத் தொடர்ந்து, மாதிரியானது 98% துல்லியத்தைக் காட்டியது. இந்த கண்டுபிடிப்புகள் IoT-இயக்கப்பட்ட ஸ்மார்ட் நகரங்களுக்கான மிகவும் பாதுகாப்பான DL-அடிப்படையிலான எதிரிடையான தாக்குதல் கண்டறிதல் அமைப்புகளை மேம்படுத்துவதற்கு பங்களிக்கின்றன.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

IDS	Intrusion Detection System
IoT	Internet of Things
DNN	Deep Neural Network
JSMA	Jacobian based Saliency Map Attack
FGSM	Fast Gradient Sign Method
BIM	Basic Iterative Method
PGD	Projected Gradient Descent
DoS	Denial of Service
ISP	Internet Service Providers
DL	Deep Learning
RDS	Random Forest-DNN and Support Vector Machine
RDL	Random Forest-DNN and Logistic Regression

RDLS	Random Forest-DNN-Support Vector Machine and Logistic Regression
CPS	Cyber-Physical System
DF	Deep Fool
GAN	Generative Adversarial Networks
DS2OS	Distributed Smart Space Orchestration System

CHAPTER 1

1.1 INTRODUCTION

The rapid growth of the IoT has brought about a surge in smart city applications, which are increasingly reliant on DL models for cyber-attack detection. However, DL models are vulnerable to adversarial attacks, in which attackers craft maliciously crafted inputs that trick the model into making incorrect predictions. This can lead to serious consequences, such as device malfunction, data leakage, operational outage, and financial loss.

Adversarial training is a technique that can improve the robustness of DL models to adversarial attacks. It involves training the model on a combination of normal and adversarial examples, forcing the model to learn to distinguish between legitimate and malicious inputs. This can significantly improve the model's ability to detect adversarial attacks. By repeatedly exposing the model to these adversarial inputs, it adapts to recognize and resist them, thereby enhancing its generalization and resilience against potential attacks in real-world scenarios. In the realm of IoT based Smart City applications, the integration of adversarial training into the development of deep learning models for cyber-attack detection is essential. It helps mitigate the risks posed by sophisticated adversaries attempting to manipulate data or breach security measures within the Smart City infrastructure. In this project, at first the training data is trained using hybrid ensemble model combining deep learning algorithm like DNN and machine learning algorithms like Random Forest, SVM and Logistic regression and predictions are made. And the best suited model is chosen for further implementations. In the next step Adversarial samples are generated using various attacks like FGSM, BIM, JSMA, PGD and DF and a new Adversarial test

data is generated and this test data is used to evaluate the robustness of the trained model to adversarial attacks. And also, the model is re-trained with the new training set that contains adversarial test data and original training data.

Adversarial training is a promising technique for improving the security of DL based cyber-attack detection systems in IoT-based smart city applications. By addressing the challenges of adversarial training and developing more efficient and effective techniques robustness and generalizability of DL models, making them less susceptible to adversarial attacks and more effective in detecting and mitigating cyber-attacks in real-world scenarios. As IoT-based smart city applications continue to grow in complexity and sophistication, adversarial training will play an increasingly important role in ensuring the security and resilience of these critical infrastructure systems.

1.2 OBJECTIVES

- To Develop a hybrid ensemble model combining deep learning algorithms (DNN) with machine learning methods (Random Forest, SVM and Logistic Regression) for intrusion detection in IoT-based smart city applications.
- To implement adversarial training techniques (FGSM, BIM, JSMA, PGD, DF) to enhance the robustness of models against adversarial attacks.
- To propose method such as adversarial retraining to enhance IDS resilience against adversarial attacks, thereby improving robustness.
- To understand the potential consequences of adversarial attacks on IoT-based smart city services to emphasize the importance of robust IDS solutions.

1.3 PROBLEM STATEMENT

IDS play a critical role in safeguarding Internet of Things (IoT) applications, particularly in the context of smart city services, by distinguishing between benign and malicious network activities. However, the increasing sophistication of cyber threats poses significant challenges to the effectiveness of IDS models. Of particular concern is the vulnerability of IDS models to adversarial attacks, wherein attackers manipulate network data to evade detection by IDS algorithms.

Despite the growing reliance on machine learning (ML) techniques for intrusion detection in IoT environments, there is a lack of comprehensive understanding regarding the susceptibility of ML-based IDS models to adversarial attacks and the potential threats for IoT-based smart city services. Current research fails to adequately address this critical gap, thereby leaving IoT infrastructures vulnerable to exploitation and compromising the integrity and security of smart city ecosystems.

Hence, the primary objective of this research is to investigate the impact of adversarial attacks on DL-ML models employed in IDS for IoT applications, particularly in the context of smart city services. By evaluating the performance of IDS models under adversarial conditions and proposing effective defense mechanisms, this study aims to enhance the resilience of IDS systems against adversarial attacks, thereby fortifying the security posture of IoT-based smart city environments.

Through empirical analysis and simulation, this research seeks to not only quantify the degradation in IDS detection accuracy caused by adversarial samples but also demonstrate the efficacy of proposed defense mechanisms, such as “adversarial

retraining", in mitigating the impact of adversarial attacks. By addressing these pressing challenges, this study aims to contribute to the advancement of knowledge in the field of IoT security and facilitate the development of robust IDS solutions capable of safeguarding smart city infrastructures against evolving cyber threats.

1.4 CHALLENGES & APPLICATIONS

- Integrating various models such as deep neural networks (DNN), random forests, support vector machines (SVM), and logistic regression into a cohesive ensemble poses challenges in terms of ensuring that the features used by each model are compatible with each other, scaling the ensemble to handle large datasets efficiently.
- Creating and testing adversarial attacks, like FGSM, BIM, JSMA, PGD, and DF, demands a thorough grasp of the weaknesses and strengths of the models involved. Crafting attacks that truly expose vulnerabilities and resemble real-world threats isn't easy.
- Applications may include enhancing security in transportation systems. The ensemble model could be employed to safeguard IoT-enabled transportation infrastructure, such as traffic monitoring systems and autonomous vehicles, from cyber threats.
- Beyond intrusion detection, the ensemble model can be repurposed for anomaly detection in various domains, such as fraud detection in financial transactions, medical diagnosis, and predictive maintenance in industrial systems.
- The proposed system can be used for Dynamic Risk Assessment, offering an approach to continuously monitor and evaluate the security status in IoT networks.

1.5 ORGANIZATION OF THESIS

The outline of the thesis is as follows:

Chapter 2: Literature Survey of related works

Chapter 3: Detailed description of the proposed system and each module.

Chapter 4: Discussion of results and analysis.

Chapter 5: The final conclusion and exploration of other future works that can be implemented.

CHAPTER 2

2.1 A Supervised IDS Architecture for Mitigating IoT Security Risks

The authors came up with a new way to make IoT devices safer in smart homes. They built a special system with three main parts: First, the system learns how each IoT device normally acts. This helps it understand when something unusual happens. Then, it looks out for bad stuff happening in the network, like harmful data trying to get in. If it spots something bad, it figures out what type of attack it is and which device it's targeting. To test how well their system works, they set up a fake smart home with eight popular IoT devices. They then pretended to attack it in different ways to see if the system could catch the attacks. It did really well, spotting normal behavior accurately (96.2%), catching harmful data (90.0%), and figuring out the type of attack (98.0%). This shows their system can help keep IoT devices safe from hackers in smart homes.

2.2 Machine Learning Approaches for IoT Security: Focus on the Physical Layer

This paper delves into the growing importance of cybersecurity in Internet of Things (IoT) applications. It examines the challenges associated with securing IoT ecosystems and explores the potential of machine learning (ML) techniques to address these challenges effectively. The survey discusses the motivations driving the adoption of ML models for enhancing IoT security, emphasizing the need for innovative solutions tailored to the unique characteristics of IoT environments. It acknowledges the accelerated pace of IoT development, which has led to increased vulnerabilities and heightened security concerns. Within the survey, a systematic classification framework for ML-based approaches is presented, categorizing them

into device-oriented and network-oriented strategies. This classification aids in understanding the diverse methodologies employed to fortify IoT devices and networks against cyber threats. Drawing upon recent research findings, the survey provides insights into the practical implementation of ML-based solutions, particularly at the physical layer of IoT networks. By referencing existing works, it offers guidance and best practices for engineers and practitioners interested in bolstering IoT security through ML-driven techniques.

2.3 Enhancing Semi-Supervised Learning with Virtual Adversarial Training

This study introduces a new method called virtual adversarial training (VAT) to help improve how computers learn from data. Instead of using labels to guide learning, VAT focuses on making sure that the computer understands the data around each point, even if it's a little bit changed. VAT is especially useful for situations where we only have a few labeled examples, known as semi-supervised learning. It's fast and doesn't need a lot of computing power. By using VAT, researchers were able to achieve better results in learning tasks on different datasets, like SVHN and CIFAR-10. Paper highlights the potential of VAT as a simple and effective way to make learning algorithms work better, especially when data is limited.

2.4 Exploring Attack Algorithms for Deep Learning-Based Intrusion Detection

This paper presents an evaluation of state-of-the-art attack algorithms within the domain of deep learning-based intrusion detection. The study investigates how these attack algorithms, initially designed to deceive deep learning-based image classifiers, perform in the context of intrusion detection. By analyzing the behavior of these attack algorithms, the study identifies different patterns in feature usage. It

highlights the practical limitations faced by adversaries in manipulating features, emphasizing the preference for attack methods that focus on a limited set of features, such as the JSMA attack. They also discovered which features are most vulnerable to attack and need more protection. Future work includes to study how easily attacks can move between different types of security systems, like neural networks and traditional ones.

2.5 Comparing Two Neural Networks for Spotting Attacks in IoT Networks

This study looks at how well two types of neural networks—Feed-forward Neural Network (FNN) and Self-normalizing Neural Network (SNN)—can detect attacks in IoT networks. With the rise of IoT, 5G, and AI in Industry 4.0, security incidents in IoT networks are increasing. Using data from the Cyber Range Lab at UNSW Canberra Cyber Center, the study compares FNN and SNN in terms of how accurately they spot attacks. FNN performs better overall for detecting attacks in IoT networks, according to metrics like accuracy and precision. However, when it comes to handling tricky situations like attacks designed to fool the networks (adversarial samples), SNN shows more resilience. This suggests that SNN could be a promising option for making IoT networks safer in the future.

2.6 Enhancing Intrusion Detection Efficiency through Feature Selection and Ensemble Approach

This research focuses on making intrusion detection (ID) systems work better to protect networks from threats like spammers and attackers. The main challenge is making these systems more efficient, as threats keep growing on the internet. The

study does three different tests. First, it builds systems using all 41 features related to intrusion detection. Then, it tries a method called feature selection to pick out the most important features using an Entropy-based analysis. Next, it uses these selected features to test different algorithms—Naive Bayes, Adaptive Boost, and PART—to see how well they work. In the third analysis, an Ensemble Approach is employed, utilizing Information Gain to select the best features from the entire set of 41 features. This subset of features is then evaluated using the same algorithms—Naive Bayes, Adaptive Boost, and PART—to compare the performance against the previous feature selection method. The outcomes of this analysis are examined to determine the effectiveness of the ensemble approach in improving intrusion detection efficiency.

2.7 Understanding Attacks and Defenses in Deep Neural Networks

This study explores how deep neural networks (DNNs) can be tricked by adversaries using sneaky inputs called adversarial samples. Despite their impressive performance in many tasks, DNNs are vulnerable to these tricks.The research introduces new methods for creating adversarial samples, using a detailed understanding of how DNNs work. In tests involving computer vision, these methods successfully fooled the DNN into making mistakes 97% of the time, while changing only a small part of the input. Additionally, the study assesses the susceptibility of different sample classes to adversarial perturbations by defining a measure of hardness. Finally, preliminary work on defense mechanisms against adversarial samples is outlined, focusing on developing predictive measures to quantify the distance between benign inputs and target classifications.

2.8 Improving Security in Collaborative Edge Computing for Social IoT: A Approach using Generative Adversarial Networks

This paper tackles the challenge of detecting intrusions in IoT networks, focusing on Cyber-Physical Systems (CPS)-enabled environments. It introduces a new method using Generative Adversarial Networks (GANs) to spot various types of attacks. The method has three main parts: extracting features from network data, creating a model to detect single attacks using GANs, and building a model to detect multiple attacks using GANs. First, the network data is preprocessed and features are extracted. Then, a detection model is made to catch single attacks using GANs. Finally, multiple attack detection models are combined within the GAN framework. Tests on the CSE-CIC-IDS2018 and CIC-DDoS2019 datasets show that this method is much better at detecting intrusions than previous ones. The article also talks about future improvements, like combining Convolutional Neural Networks (CNNs) with GANs to better analyze network data, and developing a faster way to extract features for real-time detection.

2.9 Protecting Deep Learning Models: A New Approach with AEDPL-DL Framework

This paper introduces a new way to make deep learning models safer from attacks called AEDPL-DL. It's like adding a shield to the models. The framework uses a mix of deep learning and other smart techniques to spot and block tricky inputs called adversarial examples (AEs).A key part of AEDPL-DL is the protection layer, which checks for AEs before the model makes a decision. Tests show that the framework works well on different datasets like MNIST and CIFAR-10, almost as good as when the data is clean. The framework is also good at spotting AEs, with high accuracy

rates. It's tested against different attacks and does as well as other top methods. Future plans include making the framework even better at finding AEs, using better data, trying new tricks, and making it faster.

2.10 Protecting IoT Networks from Tricky Attacks: Introducing FGMD Defense

This paper deals with the challenge of keeping IoT networks safe from sneaky attacks that can fool traditional detection methods. It introduces a new defense approach called FGMD to tackle this problem. First, it explains how to generate tricky inputs, called adversarial samples, without needing to know about the detection system. Then, it presents the FGMD approach, which defends against these tricky attacks.Tests on datasets show that the adversarial samples can make standard detection systems less effective. FGMD, however, stands strong against these attacks and performs better than standard methods. Even without adversarial samples, FGMD still works well. In fact, when compared to other defense methods, FGMD comes out on top. This makes FGMD a valuable tool for protecting IoT networks from sneaky attacks. While FGMD shows promise, there's still room for improvement. Future work will focus on making FGMD even better.

2.11 Adversarial Attack and Defense Strategies in Deep Learning for Communication Jamming

This study delves into enhancing air travel safety by developing deep learning models to detect communication jamming. However, deep learning models are susceptible to sophisticated attacks known as adversarial examples. Despite existing defense methods, their effectiveness is limited. In this research, novel methods are

proposed to improve both attack and defense strategies. A double-level attack approach is introduced, dynamically adjusting strategies and leveraging signal characteristics to create more potent adversarial examples. Additionally, the models are trained with adversarial examples and their knowledge is transferred to similar models in various communication environments through transfer learning. Simulation results validate the efficacy of these methods in bolstering air transportation security by enhancing jamming detection accuracy.

2.12 A New Intrusion Detection and Prevention System Using a Hybrid Deep Neural Network in Cloud Environment

The paper talks about how to keep cloud systems safe from cyber-attacks, which are becoming more common. They suggest using a smart system called LSTM to look at network traffic and find any weird patterns that might mean an attack is happening. Their tests show that this approach works really well, with high accuracy of 99% in spotting attacks and preventing them. This means it's a good solution for protecting cloud systems, especially when resources are limited.

2.13 Effective Intrusion Detection System using Hybrid Ensemble Method for Cloud Computing

The paper shows about how popular cloud computing is for businesses because it's flexible and cost-effective. But it also mentions the big problem of security. Hackers can easily attack cloud systems, causing a lot of damage in a short time. To deal with this, paper suggests using intrusion detection systems (IDS) to spot any suspicious activity. There are different types of IDS, but one called hybrid detection seems to work best because it combines two other methods. The paper also mentions different

ways to set up IDS, like on individual computers (HIDS) or across a whole network (NIDS).

2.14 A critical review on Intrusion Detection System in IoT based on ML Approach

The paper outlines the challenges associated with ensuring security and privacy in Internet of Things (IoT) environments despite its widespread application across various domains. It highlights the importance of Intrusion Detection Systems (IDSs) in safeguarding networks and information systems but notes that traditional IDS techniques are not suitable for IoT due to resource limitations and device heterogeneity. The paper conducts a survey focused on IDS-based studies in the context of IoT with the aim of identifying trends, addressing challenges, and uncovering future research opportunities. It discusses various classification methods for IDSs, explores different attributes, and analyzes the effectiveness of IDS methods.

2.15 An Effective Utilization of Machine Learning Algorithms in IoT Based Intrusion Detection System

The paper discusses the increasing importance of security measures, particularly in light of the growing threat of cyberattacks. It highlights the role of Intrusion Detection Systems (IDS) in protecting internet users and businesses from malicious network activity. Also focuses on the integration of Machine Learning (ML) algorithms within IDS to enhance threat detection and classification. Specifically, it compares the effectiveness of different ML algorithms, such as Classification and Regression Trees (CART), Linear Discriminant Analysis (LDA), and Random

Forest, in various applications including IoT, cloud services, big data, 5G networks, and smart cities. Using the KDD-CUP dataset, the study validates a proposed model and analyzes its performance compared to other methodologies.

2.16 Hybrid Intrusion Detection System Using Machine Learning Algorithm

The paper discusses the utilization of the Random Forest (RF) algorithm within a Hybrid Intrusion Detection System (HIDS) to develop intrusion patterns based on training data. Through the integration of an outlier detection mechanism, anomalies are identified as potential intrusions. This hybrid detection system combines the strengths of both anomaly and misuse detection methods to improve overall detection performance. The paper examines the performance of two hybrid approaches: K-means clustering with RF classifier and Gaussian mixture clustering with RF classifier, focusing on intrusion detection. The proposed framework undergoes evaluation using the NSL-KDD dataset. Results indicate notable reductions in both feature set size (by 20%) and required training sample size (up to 80%) compared to previous models. The study introduces an innovative hybrid ML-based NIDS framework aimed at simplifying computational complexity while preserving high intrusion detection performance.

2.17 A New High-Performance Feature Selection Method for Machine Learning-Based IoT Intrusion Detection

The paper discusses the challenges faced by IoT networks due to increasing data traffic and cyber-attacks, which require effective protection tools like Intrusion Detection Systems (IDSs). However, current IDS systems struggle to adapt to the diverse nature of IoT environments. To address this, the study introduces a new

method for selecting important features in IDS. They use two approaches, Gain Ratio (GR) and Information Gain (IG), to find the best features and then extract them using mathematical set theory. Authors tested this method using different ML techniques on IoT intrusion datasets. Results show that their method works well, achieving high accuracy in identifying relevant features. It outperforms other techniques, reaching the highest classification accuracy of 89.70%.

2.18 Enhancing Attack Detection and Explanation in IoT Environments Using Ensemble Blending Models

The paper discusses the challenge of securing the Internet of Things (IoT) against cyber threats and emphasizes the crucial role of Intrusion Detection Systems (IDS). However, it highlights a significant concern: the lack of clear explanations for IDS decisions. In response, the paper introduces a new approach that combines a blending model for attack classification with counterfactual and Local Interpretable Model-Agnostic Explanations (LIME) techniques to enhance explanations. They conducted experiments using real-time and large-scale benchmark datasets for IoT attacks to assess the effectiveness of their approach. Results show significant improvements in attack detection accuracy compared to traditional IDS methods. Additionally, their approach provides clear and interpretable insights into classification decisions, enabling users to make informed security choices.

2.19 IoT Network Cybersecurity Assessment with the Associated Random Neural Network

The paper introduces a new method called the Associated Random Neural Network (ARNN) to check the security of IoT networks. It's designed to detect compromised

devices by learning from real attack data. ARNN works by analyzing incoming traffic and understanding how devices in the network are connected. It's especially useful for networks with a few hundred devices, like private or industrial networks. Unlike traditional methods that need separate detectors for each device, ARNN uses one network to evaluate all devices together, which saves on computing power. The paper discusses how ARNN learns and how well it performs compared to other methods. Overall, ARNN shows promise for accurately spotting compromised devices, but it might take a while to train.

2.20 IoT-AD: A Framework to Detect Anomalies among Interconnected IoT Devices

The paper addresses the challenge of abnormal behavior in IoT devices affecting others and introduces IoT Anomaly Detector (IoT-AD) to address it. IoT-AD not only identifies anomalies but also contains their impact on other devices. It was tested using simulated and real-world data, proving its accuracy and speed in detecting anomalies. IoT-AD's effectiveness was compared with existing methods, highlighting its superiority. The framework offers a comprehensive solution for managing anomalies in IoT environments, ensuring smooth operations. Its implementation can prevent disruptions caused by abnormal behavior, enhancing IoT system reliability. Overall, IoT-AD represents a significant advancement in IoT security and performance management.

2.21 SUMMARY OF RELATED WORKS

The related works explores a broad range of topics within the realm of cybersecurity, particularly focusing on Intrusion Detection Systems (IDS) and security measures in Internet of Things (IoT) environments. Various innovative approaches are explored to enhance attack detection, prevention, and explanation in diverse contexts. One notable study proposes a supervised IDS architecture tailored for IoT security, demonstrating impressive accuracy rates in identifying anomalous behavior and potential attacks within smart home environments. Machine learning approaches are extensively investigated for IoT security, with surveys emphasizing the importance of tailored solutions to combat evolving cyber threats. Additionally, techniques like virtual adversarial training and deep learning-based intrusion detection are introduced, showcasing their efficacy in bolstering cybersecurity measures.

Hybrid intrusion detection systems and feature selection methods are also explored to improve the efficiency and performance of IDS in detecting cyber threats. Ensemble blending models emerge as a promising approach for enhancing attack detection and explanation in IoT environments, offering clear insights into classification decisions and empowering users to make informed security choices. Furthermore, advancements in securing IoT networks from adversarial attacks are highlighted, with studies introducing novel defense mechanisms like FGMD and AEDPL-DL. These approaches aim to mitigate the vulnerabilities of deep learning models and traditional detection methods against sophisticated attacks.

CHAPTER 3

3.1 ARCHITECTURE DIAGRAM

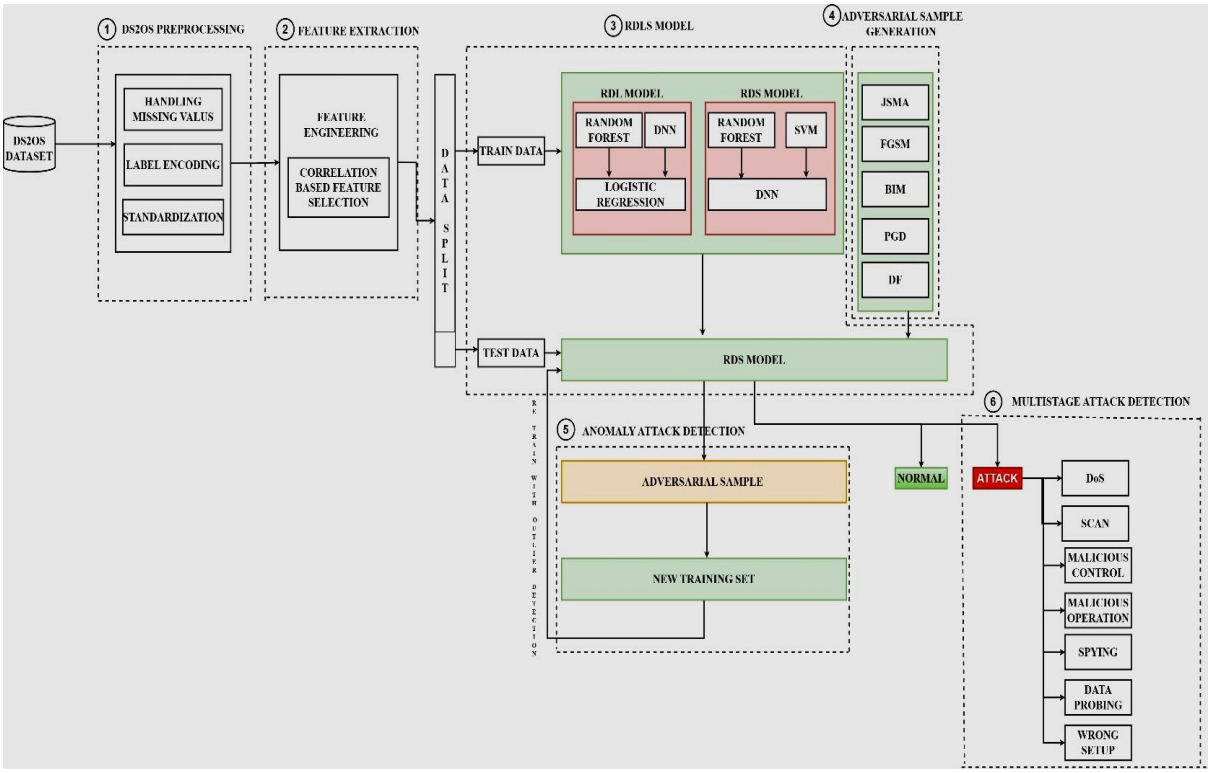


Figure 3.1: Proposed Multistage IDS Architecture Diagram

Figure 3.1 depicts the architectural diagram proposed for the system, providing an overview of its structural design.

3.2 PROPOSED SYSTEM

To ensure data quality and compatibility with hybrid models for IoT datasets, missing values are handled through imputation techniques, label encoding is applied for categorical variables, and standardization is performed for numerical features. The preprocessed dataset is then split into 80% training and 20% testing sets for

model evaluation. Random Forest, Deep Neural Network, and Logistic Regression models are trained as part of RDL, while Random Forest, Deep Neural Network, and Support Vector Machine models constitute RDS, all trained on the training data. The trained RDS model is utilized to predict accuracy on the testing data for insights into attack detection. Adversarial samples are generated from the testing data using tailored attack methods designed for hybrid models, and the resilience of the RDS model against these attacks is evaluated, considering both decision boundaries and accuracy effects. Adversarial samples are combined with the original dataset to create a balanced and integrity-maintained new training set. The RDS model is then retrained using this new dataset to enhance its robustness against adversarial attacks while maintaining accuracy. The effectiveness of the retrained RDS model in attack detection in IoT scenarios is assessed, considering accuracy, robustness, and resilience against adversarial threats.

3.3 LIST OF MODULES

The list of modules involved in the entire process is as follows:

- DS2OS Preprocessing
- Feature Extraction
- RDLS Model
- Adversarial Sample Generation
- Anomaly Attack Detection
- Multistage Attack Detection

3.4 MODULES

3.4.1 DS2OS Preprocessing:

While performing the first step of data preprocessing, null values were checked in the dataset, and label encoding was performed. Initially, an instance of the LabelEncoder class was created to encode categorical labels into numerical values. standardization of the feature variables for a machine learning model is conducted using the StandardScaler from scikit-learn as shown in figure 3.4.1.

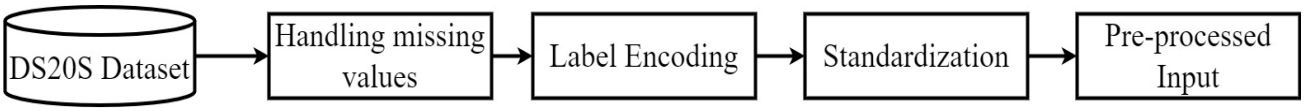


Figure 3.4.1: DS2OS Preprocessing

Algorithm:

Input: DS2OS Dataset

Output: Pre-processed data

1.Load DS2OS Dataset using pandas read_csv.

2.Handle missing values using mean

3.Handle unexpected entries by replacing specific strings with numerical values using a predefined dictionary mapping.

4.Convert all categorical features into numerical representations using Label encoding.

5.Standardize numerical features.

3.4.2 Feature Extraction:

The TimeStamp column was removed from the pre-processed data, resulting in 12 features selected for further processing as shown in figure 3.4.2.

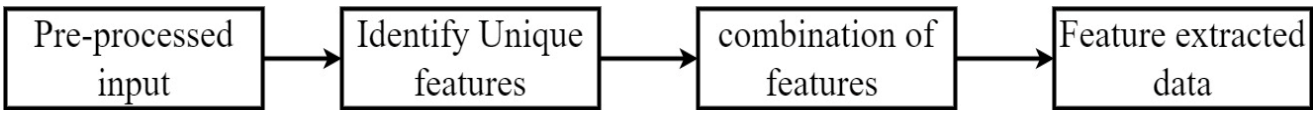


Figure 3.4.2: Generating New Feature

Algorithm:

Input: Pre-processed data

Output: Feature extracted data

1.Unique features were identified after pre-processing:

- Let X_preprocessed be the pre-processed dataset.
- Identify the unique features present in X_preprocessed after pre-processing.

2.Correlation between newly extracted features and target class were observed:

- Let y be the target class.
- For each newly extracted feature f_i in X_preprocessed:
- Compute the correlation coefficient between f_i and y.
- Evaluate the strength and direction of the correlation.

3.The newly generated feature that showed the best correlation was chosen for further analysis:

- Select the feature f_best with the highest absolute correlation coefficient with the target class y.

-Verify that the selected feature meets any additional criteria or constraints, if applicable.

4.The best five features were further taken for model training:

- Rank the remaining features based on their correlation with the target class.
- Select the top five features with the highest correlation coefficients for further analysis and model training.

Table 3.4.2.1: Feature Combination – Corresponding Extracted Feature

Features Combined	Extracted Feature
sourceLocation_code, destinationLocation_code	Location_relationship
sourceAddress_code, destinationServiceAddress_code, accessedNodeAddress_code	Address_relationship
sourceAddress_code, sourceType_code, accessedNodeAddress_code	Source_relationship

3.4.3 RDLS Model:

The dataset was then split into training and testing sets in an 80:20 ratio. Subsequently, Random Forest, SVM, DNN, and Logistic Regression models were trained and tested, followed by generating predictions from them as shown in figure 3.4.3.

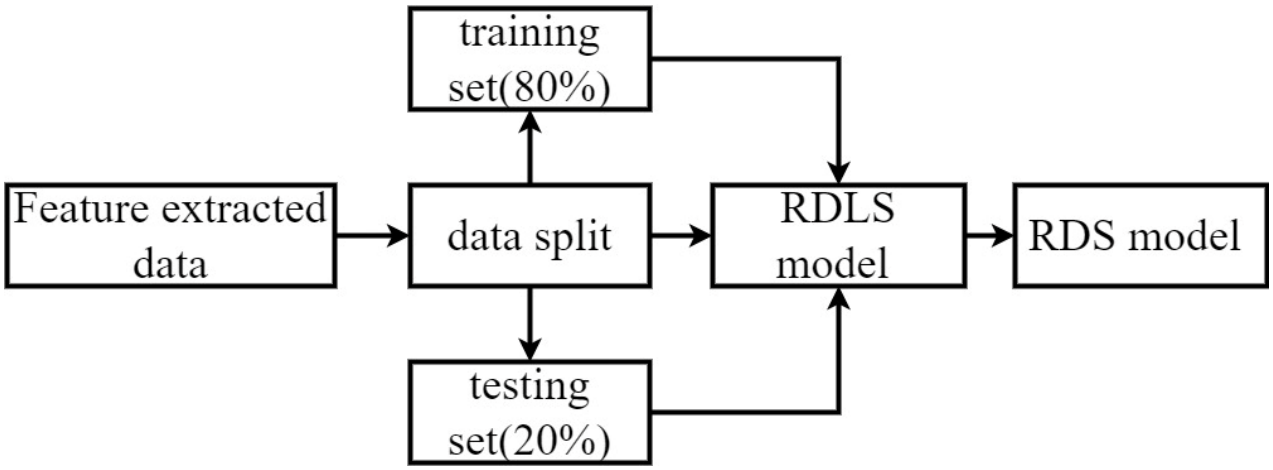


Figure 3.4.3: RDLS Model Training

Algorithm:

Input: Feature extracted data

Output: RDS Model

1. Split the data into training (80%) and testing (20%) sets using an appropriate splitting method like train_test_split from scikit-learn:

- Let X be the feature extracted data.
- Let y be the corresponding target labels.
- Use train_test_split to randomly split X and y into training and testing sets with an 80:20 ratio.

2. Train RDLS model using training data:

- For each model M in RDLS (Random Forest, SVM, DNN, Logistic Regression):
  - Train M using the training data.

3. Test the RDLS model using test data:

- For each model M in RDLS:
  - Use M to predict the target labels for the test data.

4. Evaluate the performance of both RDS and RDL models on the unseen test data using Accuracy, precision, recall, and F1-score:

- For each model M in RDLS:
  - Calculate the performance metrics using the predicted labels and the true labels:
    - Accuracy,Precision,Recall,F1-score

3.4.4 Adversarial Sample Generation:

Table 3.4.4.1: Numerical representation of Attacks

Name of the Attack	Corresponding Label
DoS	0
Data Probing	1
Malicious Control	2
Malicious Operation	3
Scan	4
Spying	5
Wrong Setup	6

Adversarial sample generation is the process of creating carefully crafted inputs that are designed to fool the deep learning model into making incorrect predictions. These adversarial examples can be used to test the robustness of the model and identify potential weaknesses.

Table 3.4.4.2: Attacks Description

Attack Technique	**Description**	**Attack Generated**						
		0	**1**	**2**	**3**	**4**	**5**	**6**
FGSM	Adds a small, quick change to an input to fool the model					✓	✓	✓
JSMA	Alternate data to confuse a neural network's predictions		✓		✓	✓		
BIM	Takes multiple small steps to change an input, making it harder for the model to catch on		✓		✓	✓		
PGD	This is a more advanced version of FGSM. It iteratively refines the input data with small modifications to maximize the model's error.					✓	✓	✓
DF	Finds the smallest possible change to an input needed to fool the model	✓	✓		✓	✓		

This module is used for improving the robustness of deep learning models to adversarial attacks. This is done by training the model on a combination to learn to distinguish between legitimate and malicious inputs, making it more difficult to fool as shown in figure 3.4.4.2.

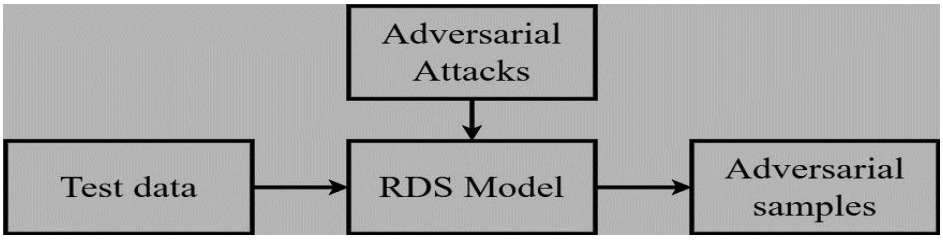


Figure 3.4.4.2: Adversarial Sample Generation

Algorithm:

Input: Test data

Output: Adversarial Samples

1. Adversarial sample generation:

   -Choose attack methods.

   -Generate perturbation based on the attack methods.

   -Apply perturbation to original input data to create adversarial example.

2. Test the RDS model with generated adversarial samples.

3.4.5 Anomaly Attack Detection:

When generating adversarial samples, there is a possibility of encountering unknown attacks, referred to as anomaly attacks, in addition to signature attacks. If anomaly attacks are detected, the model should be retrained using a new training set obtained from adversarial samples, employing outlier detection techniques to eliminate anomalous samples as shown in figure 3.4.5.

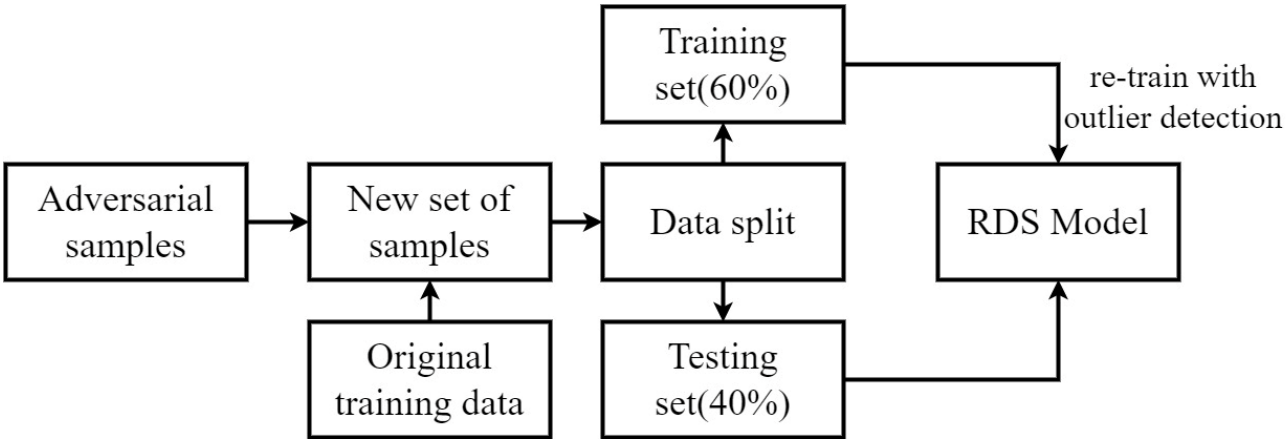


Figure 3.4.5: Anomaly Attack Detection

Table 3.4.5.1: Train-Test Samples

	Before Generating Adversarial Samples	After Generating Adversarial Samples
Train	2,86,352	2,93,852
Test	71,589	71,589

Algorithm:

Input: Adversarial samples

Output: Retrained RDS Model

1. Generate adversarial samples using attack methods:

Let adversarial_samples be the generated adversarial samples using attack methods.

2. Apply an outlier detection technique to identify anomaly attacks:

Let outlier_detection_technique be the selected technique for identifying anomaly attacks.

Apply(outlier_detection_technique, adversarial_samples) -> Identify anomaly attacks.

3. New Training and Testing Set Construction:

- Combine original training set and correctly classified adversarial samples:

Let original_training_set be the original training set.

Let correctly_classified_adversarial_samples be the adversarial samples correctly classified by the RDS model.

Combine(original_training_set, correctly_classified_adversarial_samples) -> New training samples.

- Split the data into new train (60%) and test (40%) sets:

Split(new_training_samples, test_set) -> New training set (60%) and test set (40%).

4. Retrain the model using the new training set:

Let retrained_RDS_model be the model retrained using the new training set.

Train(retrained_RDS_model, new_training_set).

5. Assess the retrained model's performance on a test set:

Test(retrained_RDS_model, test_set) -> Performance evaluation.

3.4.6 Multistage Attack Detection:

In this stage for model retraining, use new training data that includes both original data and adversarial samples. Compare the predictions from the retrained model with its original predictions to understand how adversarial inputs affect its performance. And a classification model is used to classify the seven different classes of attacks as shown in figure 3.4.6.

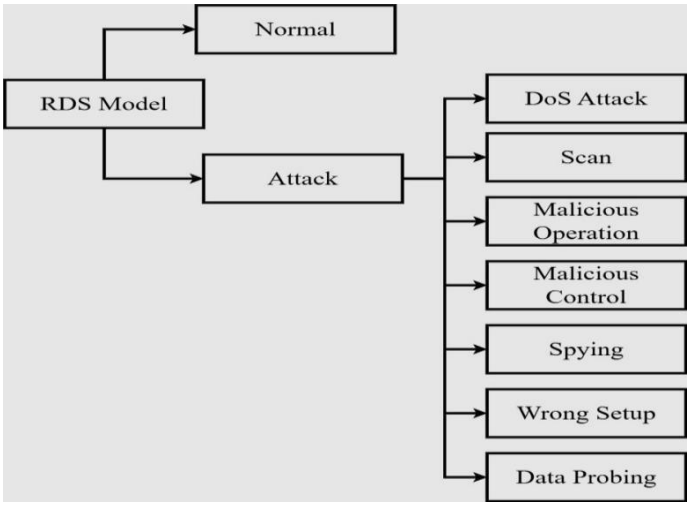


Figure 3.4.6: Multistage Attack Detection

Algorithm:

Input: Retrained RDS Model

Output: Classification of attacks

1. Combine the original clean training data with generated adversarial samples, creating a new training set:

Let original_training_data be the original clean training data.

Let adversarial_samples be the generated adversarial samples.

Combine(original_training_data, adversarial_samples) -> New training set.

2. Retrain Model:

Train the model again on this new training set, incorporating the knowledge of

adversarial examples:

Let retrained_RDS_model be the model retrained using the new training set.

Train(retrained_RDS_model, new_training_set).

3. Evaluation:

- Predict Both Types:

Make predictions on both normal data and unseen attack samples:

Let test_data be the unseen attack samples.

Predict(retrained_RDS_model, test_data) -> Predictions for both normal data and unseen attack samples.

- Monitor Performance:

Evaluate the model's performance on each type of data to understand how retraining with adversarial examples impacts its ability to handle attacks:

EvaluatePerformance(retrained_RDS_model, normal_data) -> Performance metrics for normal data.

EvaluatePerformance(retrained_RDS_model, attack_samples) -> Performance metrics for attack samples.

CHAPTER 4

4.1 DATASET DESCRIPTION

- The dataset is derived from a Smart Space simulation consisting of seven interconnected micro services provided by various IoT devices.
- The simulated IoT devices include light controllers, movement sensors, thermostats, solar batteries, washing machines, door locks, and user Smartphone's.
- The dataset encompasses seven distinct attack scenarios such as Net Scan, Spying, Malicious Control, Malicious Operation, DoS, Data probing, wrong setup.
- The DS2OS dataset comprises a total of 10,027 anomalous samples, representing attack scenarios, and 347,935 normal class samples, capturing regular IoT device behavior.
- Each sample in the dataset contains 13 features they are

Table 4.1: Dataset Description

Feature	**Type**		**Description**
	Categorical	Numerical	
Source ID	✓		Source μS, e.g., solar_batteries100, door_locks242
Source Address	✓		Address of the μS, e.g., agent61 solar_batteries100
Source Type	✓		Seven services
Source Location	✓		21 service locations including rooms and other locations, e.g., garage and entrance.

Destination Service Address	✓		Similar to source address
Destination Service Type	✓		Seven services
Destination Location	✓		21 service locations including rooms and other locations, e.g., garage and entrance.
Accessed Node Address	✓		Similar to source address,
Accessed Node Type	✓		e.g., agent21 door_locks79
Operation	✓		Read, Write, Register Service, and Lock Sub Tree
Value		✓	Value of the continuous data
Timestamp		✓	Timestamp of the packet generation
Normality	✓		Eight classes normal and seven attack types

Sample Data:

sourceID	sourceAddress	sourceType	sourceLocation	destinationServiceAddress	destinationServiceType	destinationLocation	accessedNodeAddress	accessedNodeType
lightcontrol2	/agent2/lightcontrol2	/lightControler	BedroomParents	/agent2/lightcontrol2	/lightControler	BedroomParents	/agent2/lightcontrol2	/lightControler
lightcontrol3	/agent3/lightcontrol3	/lightControler	Dinningroom	/agent3/lightcontrol3	/lightControler	Dinningroom	/agent3/lightcontrol3	/lightControler
lightcontrol1	/agent1/lightcontrol1	/lightControler	BedroomChildren	/agent1/lightcontrol1	/lightControler	BedroomChildren	/agent1/lightcontrol1	/lightControler
lightcontrol4	/agent4/lightcontrol4	/lightControler	Kitchen	/agent4/lightcontrol4	/lightControler	Kitchen	/agent4/lightcontrol4	/lightControler
movement4	/agent4/movement4	/movementSensor	Kitchen	/agent4/movement4	/movementSensor	Kitchen	/agent4/movement4	/movementSensor

Figure 4.1.1: DS2OS Dataset Samples

Figure 4.1.1 displays the original DS2OS samples comprising a total of 13 features. The simulated IoT devices encompass light controllers, movement sensors, thermostats, solar batteries, washing machines, door locks, and user smartphones.

```
normal                              347935
anomalous(DoSattack)                  5780
anomalous(scan)                       1547
anomalous(malitiousControl)            889
anomalous(malitiousOperation)          805
anomalous(spying)                      532
anomalous(dataProbing)                 342
anomalous(wrongSetUp)                  122
Name: normality, dtype: int64
```

Figure 4.1.2: Attack Class and Number of samples

Figure 4.1.2 depicts the count of normal samples attack samples alongside their respective attack types.

4.2 EXPERIMENTAL RESULTS

Platform used: Google collab

Pre-processing:

```
none_rep_data = na_rep_data.replace(['none', 'true', 'false', '0', 'False', 'True'], [0, 1, 0, 0, 0, 1])
none_rep_data
#none, true, false replaced by 0, 1, 0

none_rep_data[none_rep_data['value'] == 'twenty']

rep_twenty = none_rep_data.replace('twenty', 20)

rep_twenty[rep_twenty['value'] == 'twenty']

# replaced twenty with 20
```

```
na_rep_data = df.fillna(0)
print('null values after filling na:\n')
na_rep_data.isnull().sum()
```

```
# encoded categorical values to numerical values

# val_float.head(5)

val_float['sourceAddress_code'] = le.fit_transform(val_float['sourceAddress'])
val_float['sourceType_code'] = le.fit_transform(val_float['sourceType'])
val_float['sourceLocation_code'] = le.fit_transform(val_float['sourceLocation'])
val_float['destinationServiceAddress_code'] = le.fit_transform(val_float['destinationServiceAddress'])
val_float['destinationServiceType_code'] = le.fit_transform(val_float['destinationServiceType'])
val_float['destinationLocation_code'] = le.fit_transform(val_float['destinationLocation'])
val_float['accessedNodeAddress_code'] = le.fit_transform(val_float['accessedNodeAddress'])
val_float['accessedNodeType_code'] = le.fit_transform(val_float['accessedNodeType'])
val_float['operation_code'] = le.fit_transform(val_float['operation'])
val_float['normality_code'] = le.fit_transform(val_float['normality'])


encoded_data = val_float

encoded_data.head(10)
encoded_data.columns

pp_data.drop(columns=['timestamp'], inplace=True)
```

Figure 4.2.1: Preprocessing Data

Figure 4.2.1 showcases a sequence of steps undertaken in the data preprocessing phase. Initially, missing values are addressed by employing the mean. Subsequently, any unexpected entries are managed by substituting specific strings with numerical values, facilitated by a predefined dictionary mapping. Following this, categorical features are transformed into numerical representations utilizing Label encoding. Finally, numerical features are standardized to ensure uniformity and comparability across the dataset.

sourceID_code	sourceAddress_code	sourceType_code	sourceLocation_code	destinationServiceAddress_code	destinationServiceType_code	destinationLocation_code	accessedNodeAddress_code
21	30	2	3	30	2	3	59
32	70	2	4	68	2	4	134
15	1	2	2	0	2	2	0
33	77	2	7	73	2	7	144
55	78	3	7	74	3	7	146
...	...	...	...	...	...	...	...
25	46	2	15	46	4	15	92

Figure 4.2.2: Preprocessed Data

Figure 4.2.2 displays the dataset after preprocessing, showcasing the transformed data ready for subsequent analysis.

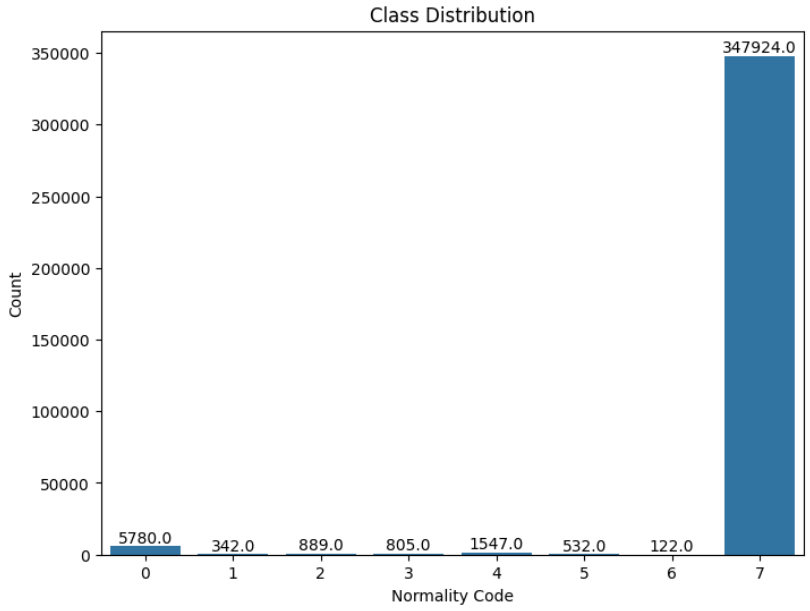


Figure 4.2.3: Class Distribution

Figure 4.2.3 displays the graph that visualizes the distribution of classes based on 'normality_code'. Each bar represents the count of instances for a particular normality code.

Feature Extraction:

```
# Feature Importance

from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt

model_extra = ExtraTreesClassifier()
model_extra.fit(ind_col,target)
print(model_extra.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model_extra.feature_importances_, index=ind_col.columns)
feat_importances.nlargest(11).plot(kind='barh')
plt.show()
```

Figure 4.2.4: Feature Importance

Figure 4.2.4 exhibits the implementation of an ExtraTreesClassifier for assessing feature importance within a dataset.

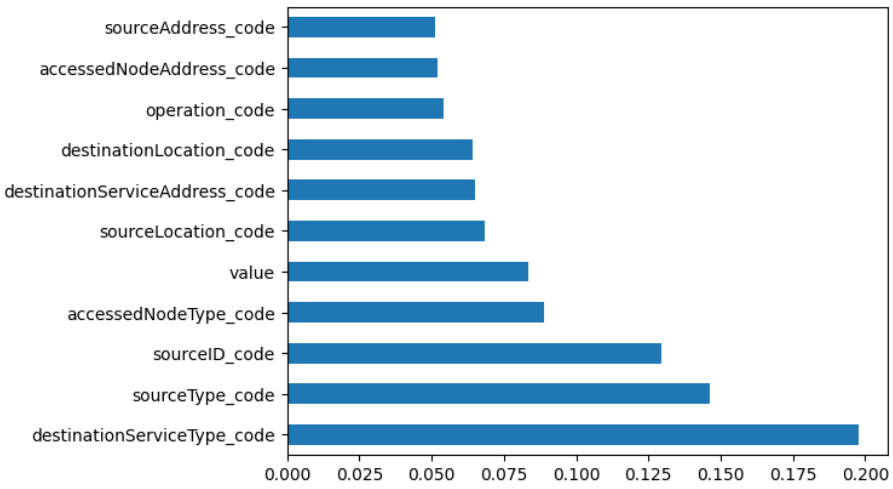


Figure 4.2.5: Feature Importance Graph

Figure 4.2.5 presents feature significance via a horizontal bar plot, facilitating identification of key contributors to the model's predictive power. Features are listed on the y-axis, with their importance scores represented on the x-axis.

```python
#combination - 1
import pandas as pd

# Create a copy of the original DataFrame
extracted_data = data.copy()

# Convert the Unicode string column to int64
data['destinationLocation_code'] = data['destinationLocation_code'].astype('int64')

# Perform the addition operation
extracted_data['location_relationship'] = data['sourceLocation_code'] + data['destinationLocation_code']


# Save the modified DataFrame to a new CSV file
extracted_data.to_csv('feature_extracted.csv', index=False)

print("Features extracted successfully.")

import pandas as pd
# Read the CSV file into a DataFrame
fe = pd.read_csv('/content/feature_extracted.csv')
fe
```

Figure 4.2.6: New Feature Extraction

Figure 4.2.6 illustrates the code demonstrating the process of combining two features to create a new feature.

destinationServiceType_code	destinationLocation_code	accessedNodeAddress_code	accessedNodeType_code	operation_code	value	normality_code	location_relationship
2	3	59	6	2	0.0000	7	6
2	4	134	6	2	0.0000	7	8
2	2	0	6	2	0.0000	7	4
2	7	144	6	2	0.0000	7	14
3	7	146	7	2	0.0000	7	14

Figure 4.2.7: Sample of Extracted Feature

Figure 4.2.7 shows the newly extracted feature generated from the code provided in Figure 4.2.6.

```python
#correlation - 1
import pandas as pd

# Calculate the correlation matrix
correlation_matrix = fe[['sourceLocation_code', 'destinationLocation_code', 'location_relationship', 'normality_code']].corr()

# Extract the correlation coefficients between location_relationship and normality_code
correlation_location_relationship = correlation_matrix.loc['location_relationship', 'normality_code']

# Extract the correlation coefficients between sourceLocation_code and normality_code
correlation_source_location = correlation_matrix.loc['sourceLocation_code', 'normality_code']

# Extract the correlation coefficients between destinationLocation_code and normality_code
correlation_destination_location = correlation_matrix.loc['destinationLocation_code', 'normality_code']

print("Correlation between location_relationship and normality_code:", correlation_location_relationship)
print("Correlation between sourceLocation_code and normality_code:", correlation_source_location)
print("Correlation between destinationLocation_code and normality_code:", correlation_destination_location)
```

Figure 4.2.8: Correlation of Extracted Feature

Figure 4.2.8 displays code calculating correlation coefficients between 'location_relationship' and 'normality_code', 'sourceLocation_code' and 'normality_code', and 'destinationLocation_code' and 'normality_code', indicating their associations.

```
Correlation between location_relationship and normality_code: 0.10882495799985813
Correlation between sourceLocation_code and normality_code: 0.1040114682761716
Correlation between destinationLocation_code and normality_code: 0.11109541844622962
```

Figure 4.2.9: Correlation Value - Extracted Feature and Target Class

Figure 4.2.9 displays correlations: 'location_relationship' vs. 'normality_code' at about 0.109. Similarly, 'sourceLocation_code' and 'destinationLocation_code' correlations of approximately 0.104 and 0.111, respectively, with 'normality_code'.

```
Feature Vectors:
[[ 6.      2.      2.     21.      0.    ]
 [ 8.      2.      2.     32.      0.    ]
 [ 4.      2.      2.     15.      0.    ]
 ...
 [20.      0.      7.     82.      1.    ]
 [40.      3.      2.     30.      0.    ]
 [40.      4.      2.     30.     20.4131]]
```

Figure 4.2.10: Feature Vectors

Figure 4.2.10 displays the selected features from dataframe as feature vectors,showing each feature's value across different instances.

```
Shape of X_train: (286352, 5)
Shape of y_train: (286352,)
Shape of X_test: (71589, 5)
Shape of y_test: (71589,)
```

Figure 4.2.11: Shapes of Train and Test Data

In Figure 4.2.11, the shapes of the training and testing datasets are depicted after feature extraction.

RDS (Random Forest, DNN, SVM) Model:

```
from sklearn.svm import SVC

# Train Support Vector Machine (SVM)
svm_model = SVC()
svm_model.fit(X_train, y_train)

# Get predictions from SVM model
svm_predictions = svm_model.predict(X_test)
```

```
from sklearn.ensemble import RandomForestClassifier

# Train Random Forest
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Get predictions from Random Forest model
rf_predictions = rf_model.predict(X_test)
```

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping

# Combine predictions from SVM and Random Forest
combined_predictions = np.column_stack((svm_predictions, rf_predictions))

# Define DNN architecture
model = Sequential([
    Dense(64, activation='relu', input_shape=(combined_predictions.shape[1],)),
    Dense(32, activation='relu'),
    Dense(8, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Train the DNN with early stopping
history = model.fit(combined_predictions, y_test, epochs=100, batch_size=32, validation_split=0.2, callbacks=[early_stopping])

# Save the trained model
model.save('RDS_model.h5')
```

Figure 4.2.12: Model Training Using RDS

Figure 4.2.12 demonstrates the training of three distinct models: Support Vector Machine (SVM), Random Forest, and a Deep Neural Network (DNN) incorporating predictions from SVM and Random Forest. The DNN comprises three layers, utilizing early stopping to mitigate overfitting. Subsequently, the trained DNN model is saved as 'RDS_model.h5'.

```
Training Accuracy: 0.9930680394172668
Training Loss: 0.04602161794900894
```

Figure 4.2.13: Training Accuracy for RDS Model

In Figure 4.2.13, the training accuracy of the Deep Neural Network (DNN) model is about 99.31%, representing the percentage of correctly classified instances. The training loss, indicating the disparity between predicted and actual values, is approximately 0.046, reflecting the model's efficacy in minimizing prediction errors during training.

```
2238/2238 [==============================] - 4s 2ms/step - loss: 0.0453 - accuracy: 0.9931
Test Loss: 0.0452636182308197
Test Accuracy: 0.9931414127349854
2238/2238 [==============================] - 3s 1ms/step
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.66      0.80      1166
           1       1.00      0.67      0.80        78
           2       1.00      1.00      1.00       186
           3       1.00      0.56      0.72       165
           4       1.00      1.00      1.00       307
           5       1.00      1.00      1.00       108
           6       1.00      1.00      1.00        19
           7       0.99      1.00      1.00     69560

    accuracy                           0.99     71589
   macro avg       1.00      0.86      0.91     71589
weighted avg       0.99      0.99      0.99     71589
```

Figure 4.2.14: Classification Report for RDS Model

Figure 4.2.14 illustrates the Deep Neural Network (DNN) model's test accuracy at approximately 99.31%, indicating accurately classified instances. The classification report provides precision, recall, and F1-score for each class, including macro and weighted averages, offering insights into overall model performance.

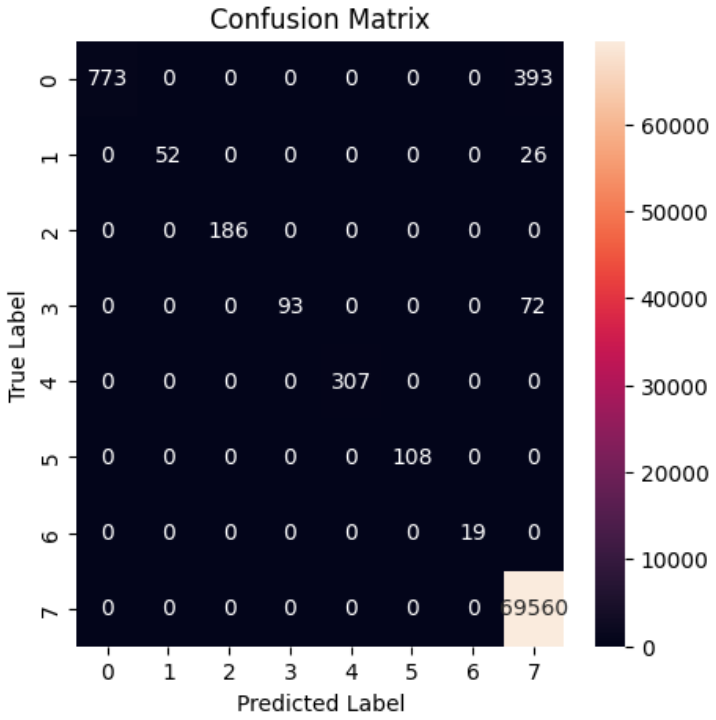


Figure 4.2.15: Confusion Matrix for RDS Model

Figure 4.2.15 illustrates the confusion matrix, depicting RDS model's performance through comparisons between actual and predicted labels.

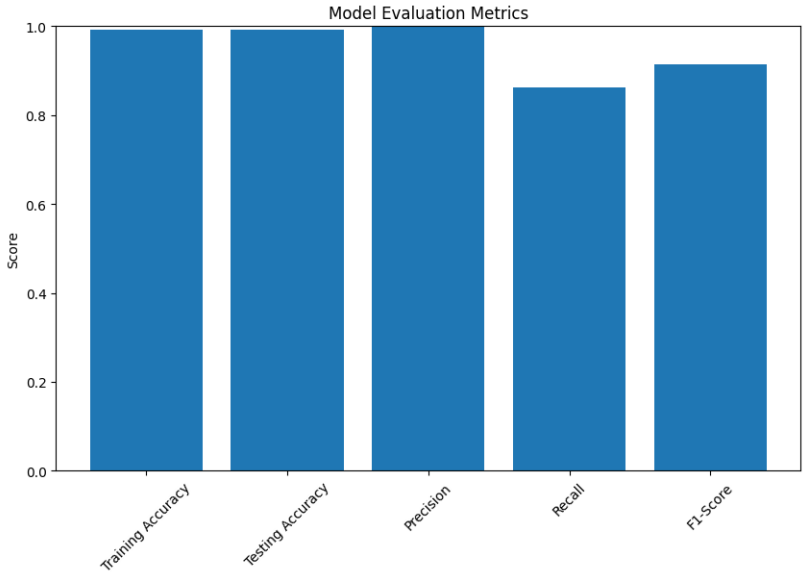


Figure 4.2.16: RDS Model Evaluation Metrics

Figure 4.2.16 shows the bar graph illustrating training accuracy, testing accuracy , precision , recall and F1-score providing a comprehensive overview of the model's performance

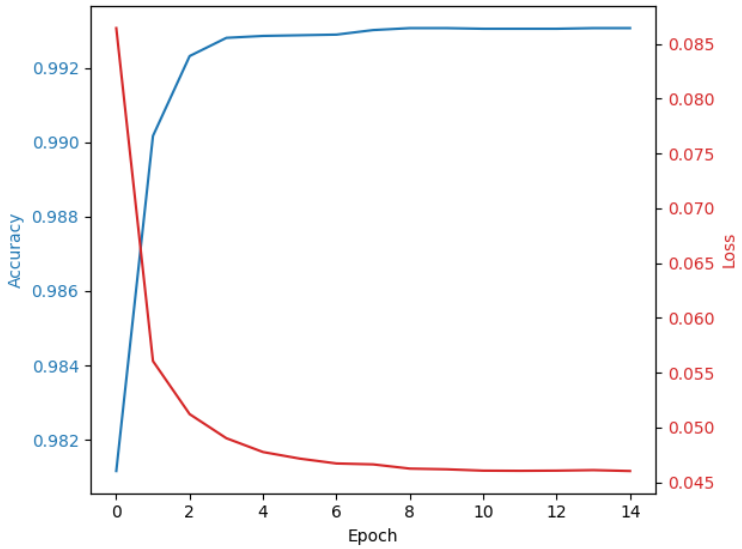


Figure 4.2.17: RDS Model Accuracy-Loss Graph

In Figure 4.2.17, the graph depicts the training accuracy and loss of the model over epochs, demonstrating their evolution during training.

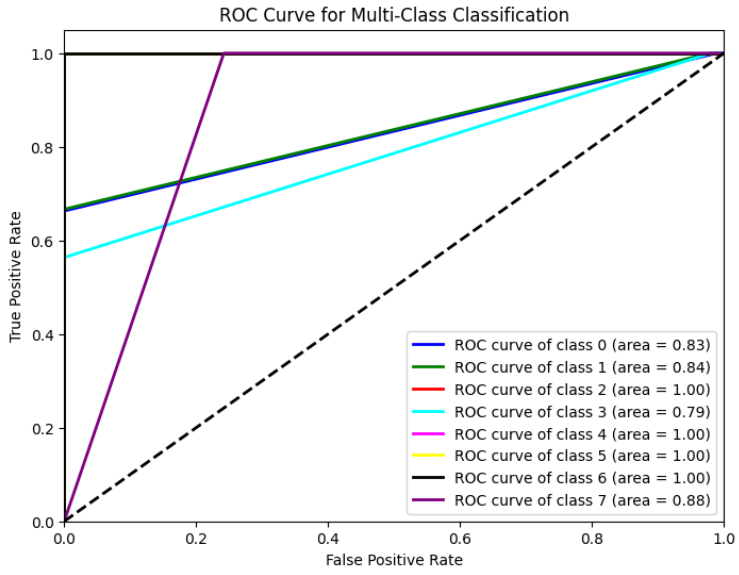


Figure 4.2.18: ROC Curve for Multi-class Classification

Figure 4.2.18 illustrates the ROC Curve for individual attack classes within the dataset, providing insights into their classification performance.

RDL (Random Forest, DNN, Logistic Regression) Model

```
from sklearn.ensemble import RandomForestClassifier

# Train Random Forest
r_model = RandomForestClassifier()
r_model.fit(X_train, y_train)

# Get predictions from Random Forest model
r_predictions = r_model.predict(X_test)
dnn_model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(8, activation='softmax')
])

# Compile the DNN model
dnn_model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

# Train the DNN model
dnn_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Get predictions from DNN model
dnn_predictions = dnn_model.predict(X_test)

from sklearn.linear_model import LogisticRegression
# Combine predictions from Random Forest and DNN
c_predictions = np.column_stack((r_predictions, dnn_predictions))

# Train Logistic Regression
logreg_model = LogisticRegression()
logreg_model.fit(c_predictions, y_test)
```

Figure 4.2.19: Model Training using RDL

Figure 4.2.19 demonstrates the training of three distinct models: Logistic regression (SVM), Random Forest, and a Deep Neural Network (DNN) incorporating predictions from DNN and Random Forest.

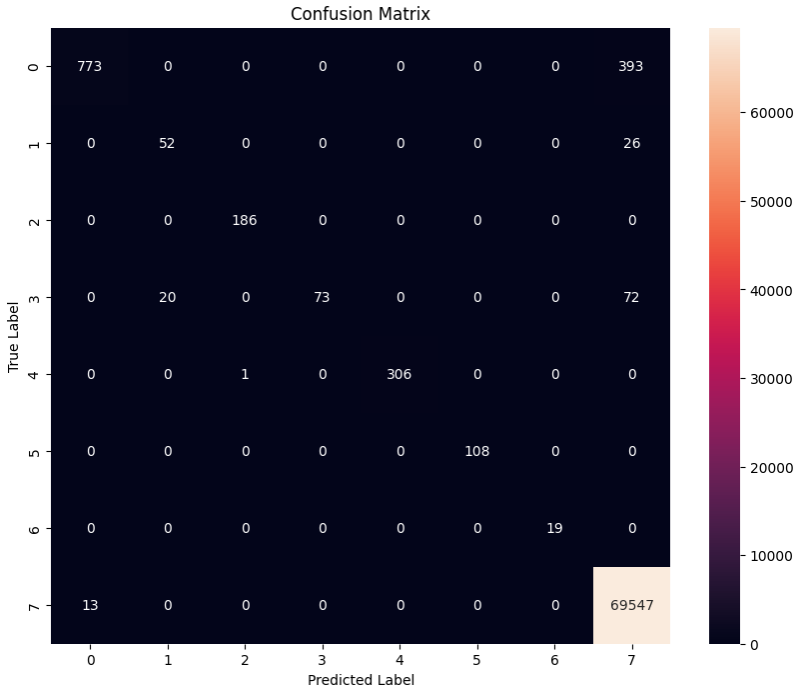


Figure 4.2.20: Confusion Matrix for RDL Model

Figure 4.2.20 illustrates the confusion matrix, depicting RDL model's performance through comparisons between actual and predicted labels.

```
Test Accuracy: 0.9926664711058961
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.66      0.79      1166
           1       0.72      0.67      0.69        78
           2       0.99      1.00      1.00       186
           3       1.00      0.44      0.61       165
           4       1.00      1.00      1.00       307
           5       1.00      1.00      1.00       108
           6       1.00      1.00      1.00        19
           7       0.99      1.00      1.00     69560

    accuracy                           0.99     71589
   macro avg       0.96      0.85      0.89     71589
weighted avg       0.99      0.99      0.99     71589
```

Figure 4.2.21: Classification Report for RDL Model

Figure 4.2.21 shows the classification report for the RDL model, offering detailed performance metrics and insights into its classification accuracy.

```
Classification Report:
              precision    recall  f1-score   support

           4       0.00      0.00      0.00         2
           5       1.00      1.00      1.00        26
           6       0.99      1.00      1.00       272

    accuracy                           0.99       300
   macro avg       0.66      0.67      0.67       300
weighted avg       0.99      0.99      0.99       300
```

Figure 4.2.22: FGSM Classification Report

Figure 4.2.21 shows the classification report for the FGSM attack technique, offering detailed performance metrics and insights into its classification accuracy.

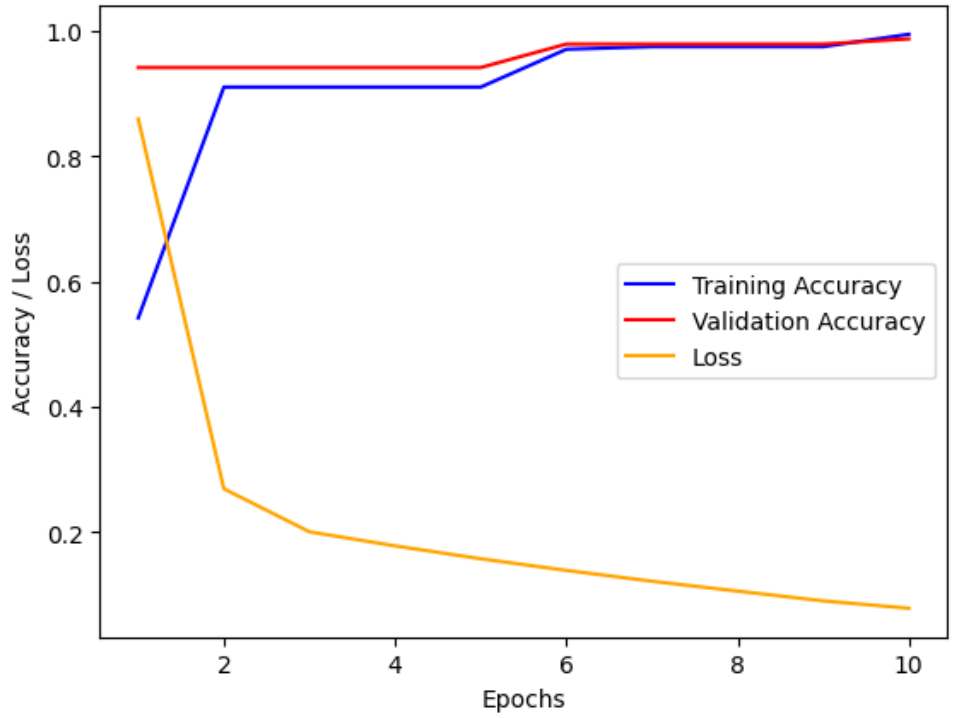


Figure 4.2.23: Accuracy-Loss Graph for FGSM

In Figure 4.2.23, the graph depicts the training accuracy, validation accuracy and loss of the model over epochs, demonstrating their evolution during training.

```
Classification Report for JSMA:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         8
           3       0.94      1.00      0.97       144
           4       0.95      0.94      0.94       148

    accuracy                           0.94       300
   macro avg       0.63      0.65      0.64       300
weighted avg       0.92      0.94      0.93       300
```

Figure 4.2.24: JSMA Classification Report

Figure 4.2.24 shows the classification report for the JSMA attack technique, offering detailed performance metrics and insights into its classification accuracy.

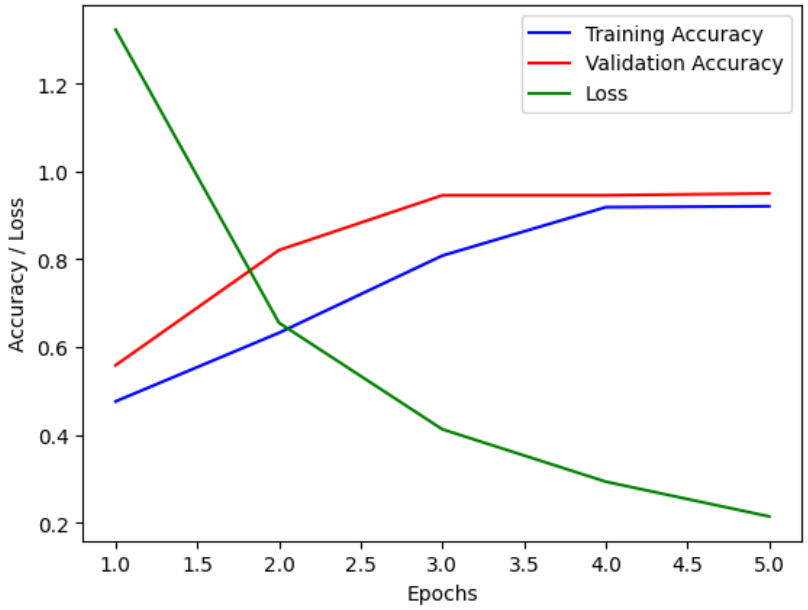


Figure 4.2.25: Accuracy-Loss Graph for JSMA

In Figure 4.2.25, the graph depicts the training accuracy, validation accuracy and loss of the model over epochs, demonstrating their evolution during training.

```
Classification Report for DF:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        87
           1       1.00      1.00      1.00       137
           3       1.00      1.00      1.00        29
           4       1.00      0.94      0.97        47

    accuracy                           0.99       300
   macro avg       0.99      0.98      0.99       300
weighted avg       0.99      0.99      0.99       300
```

Figure 4.2.26: DF Classification Report

Figure 4.2.26 shows the classification report for the DF attack technique, offering detailed performance metrics and insights into its classification accuracy.

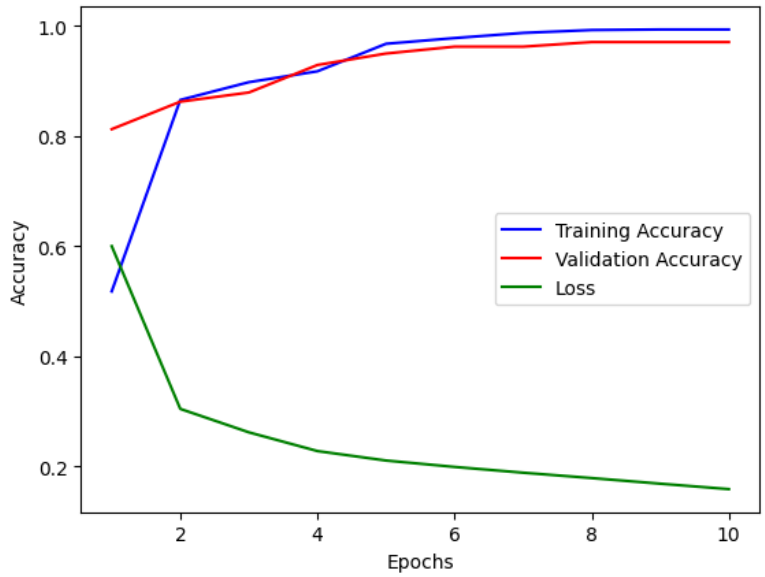


Figure 4.2.27: Accuracy-Loss Graph for DF

In Figure 4.2.27, the graph depicts the training accuracy, validation accuracy and loss of the model over epochs, demonstrating their evolution during training.

```
Classification Report for BIM:
              precision    recall  f1-score   support

           1       0.61      1.00      0.75        20
           3       0.95      0.98      0.97       151
           4       0.97      0.84      0.90       129

    accuracy                           0.92       300
   macro avg       0.84      0.94      0.88       300
weighted avg       0.94      0.92      0.93       300
```

Figure 4.2.28: BIM Classification Report

Figure 4.2.28 shows the classification report for the BIM attack technique, offering detailed performance metrics and insights into its classification accuracy.

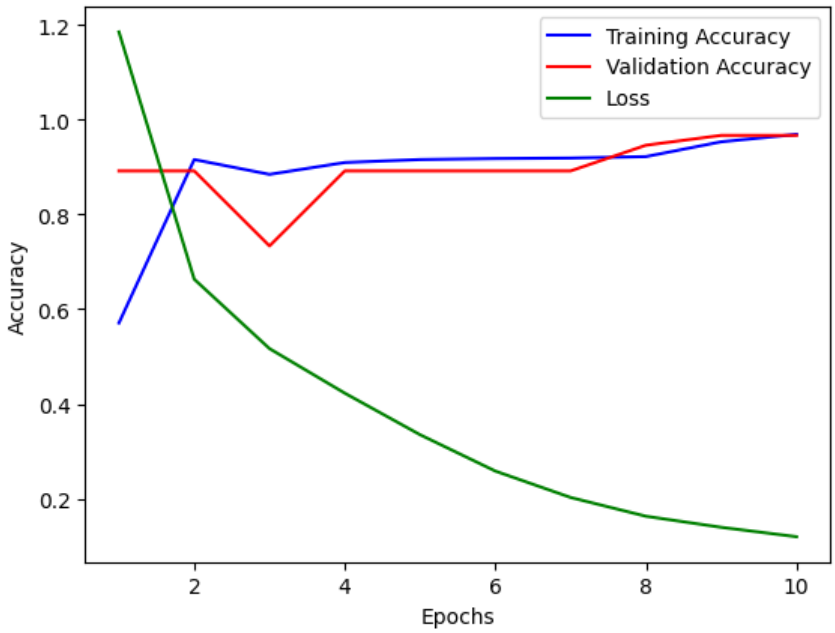


Figure 4.2.29: Accuracy-Loss Graph for BIM

In Figure 4.2.29, the graph depicts the training accuracy, validation accuracy and loss of the model over epochs, demonstrating their evolution during training.

Classification Report for PGD:

	precision	recall	f1-score	support
4	0.00	0.00	0.00	5
5	0.00	0.00	0.00	27
6	0.89	1.00	0.94	268
accuracy			0.89	300
macro avg	0.30	0.33	0.31	300
weighted avg	0.80	0.89	0.84	300

Figure 4.2.30: PGD Classification Report

Figure 4.2.30 shows the classification report for the PGD attack technique, offering detailed performance metrics and insights into its classification accuracy.

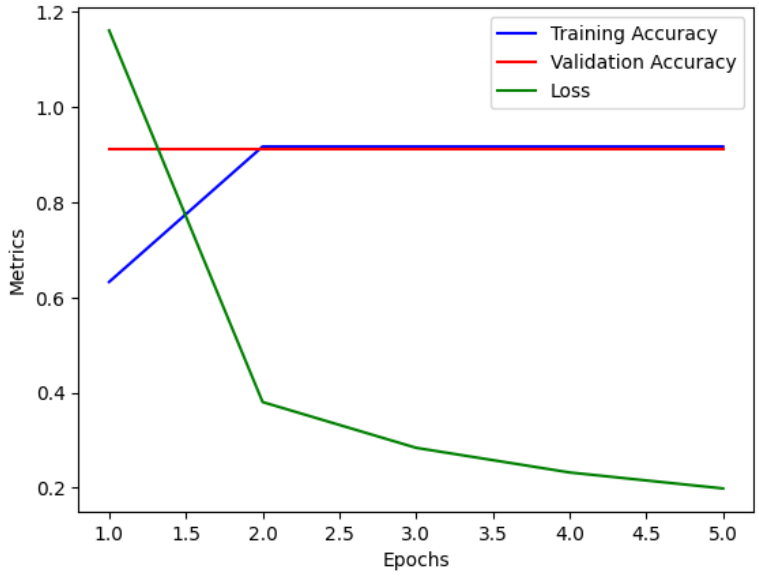


Figure 4.2.31: Accuracy-Loss Graph for PGD

In Figure 4.2.31, the graph depicts the training accuracy, validation accuracy and loss of the model over epochs, demonstrating their evolution during training.

```
# Concatenate the two datasets along the rows
adversarial = pd.concat([j,f,p,b,d], ignore_index=True)

# Save the mixed dataset
adversarial.to_csv("adversarial.csv", index=False)
m1=pd.read_csv('/content/adversarial.csv')
```

Figure 4.2.32: Concatenating Attack Samples

Figure 4.2.32 illustrates the merging of all attack samples into a unified dataset named adversarial samples.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        97
           1       0.95      0.73      0.82       146
           3       0.52      0.99      0.68       316
           4       0.55      0.46      0.50       339
           5       0.54      0.77      0.64        53
           6       0.47      0.28      0.35       549

    accuracy                           0.58      1500
   macro avg       0.67      0.71      0.67      1500
weighted avg       0.58      0.58      0.55      1500
```

Figure 4.2.33: Classification Report of Adversarial Test

Figure 4.2.33 shows the classification report for the Adversarial test data, offering detailed performance metrics and insights into its classification accuracy.

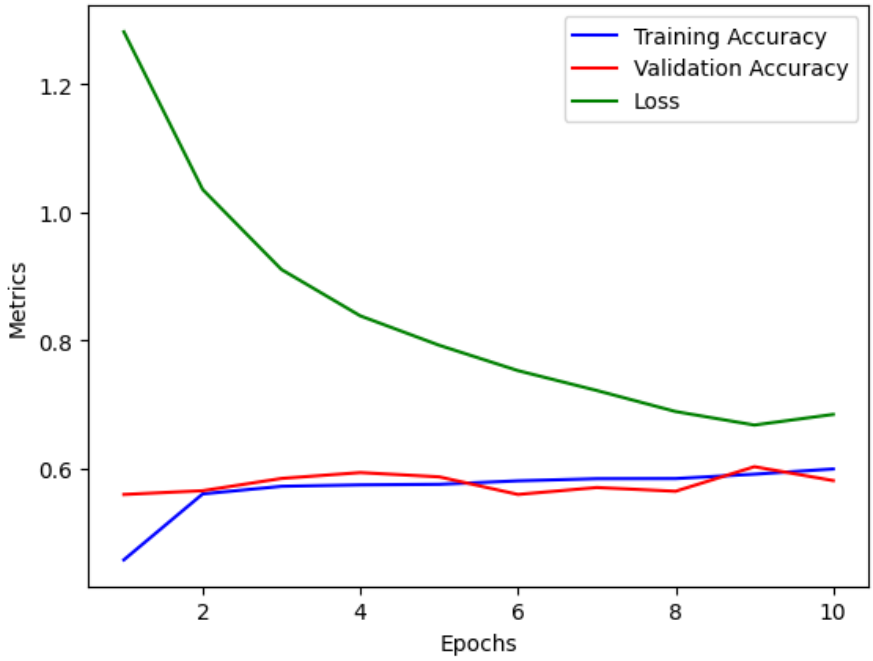


Figure 4.2.34: Accuracy-Loss Graph for Adversarial Samples

In Figure 4.2.34, the graph depicts the training accuracy, validation accuracy and loss of the model over epochs, demonstrating their evolution during training.

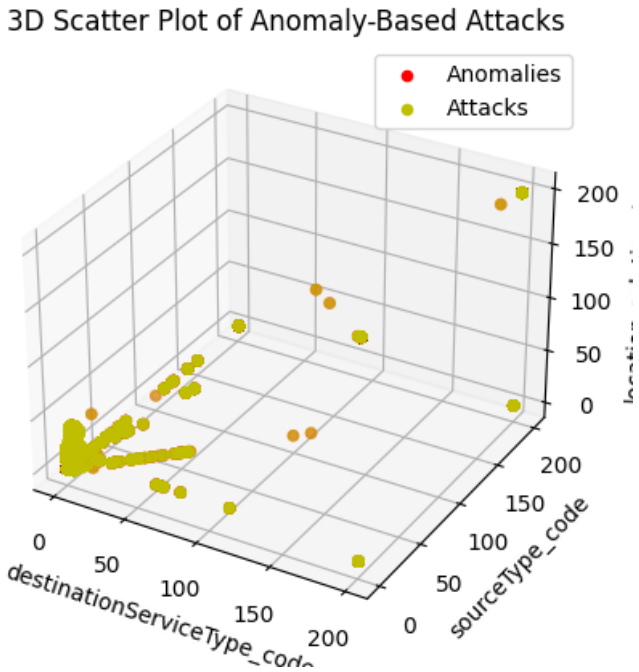


Figure 4.2.35: 3D Scatter Plot of Anomaly-Based Attacks

Figure 4.2.35 illustrates a 3D scatter plot depicting Anomaly-Based Attacks, showcasing regular attack samples alongside anomalies.

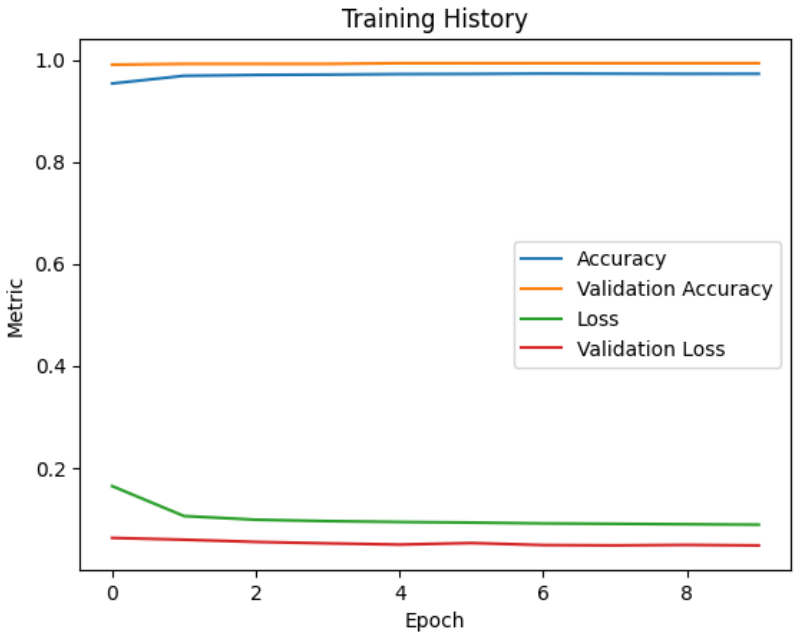


Figure 4.2.36: Training History of Re-trained Model

In Figure 4.2.36, the graph depicts the training accuracy, validation accuracy, validation loss and loss of the model over epochs, demonstrating their evolution during training.

```
Re-Training Accuracy: 0.9725159406661987
Re-Training Loss: 0.0883912742137909
```

Figure 4.2.37: Re-training Accuracy

In Figure 4.2.37, the training accuracy of the re-trained RDS model is about 97.25%, representing the percentage of correctly classified instances.

```
2331/2331 [==============================] - 6s 2ms/step - loss: 0.0791 - accuracy: 0.9775
Test Loss: 0.07912341505289078
Test Accuracy: 0.9775301814079285
2331/2331 [==============================] - 4s 2ms/step
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.71      0.83      1348
           1       0.88      0.90      0.89       389
           2       1.00      1.00      1.00       186
           3       1.00      0.19      0.32       794
           4       0.68      0.80      0.73       986
           5       0.79      0.87      0.82       203
           6       0.51      0.67      0.58      1123
           7       0.99      1.00      1.00     69560

    accuracy                           0.98     74589
   macro avg       0.86      0.77      0.77     74589
weighted avg       0.98      0.98      0.98     74589
```

Figure 4.2.38: Re-training Classification Report

Figure 4.2.38 shows the classification report after re-training, offering detailed performance metrics and insights into its classification accuracy.

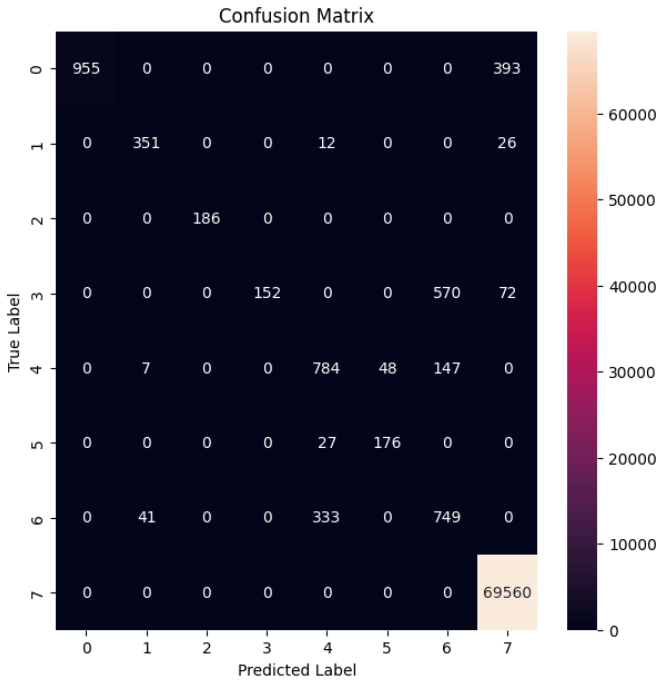


Figure 4.2.39: Re-training Confusion Matrix

Figure 4.2.39 illustrates the confusion matrix, depicting RDS model's performance after re-training through comparisons between actual and predicted labels.

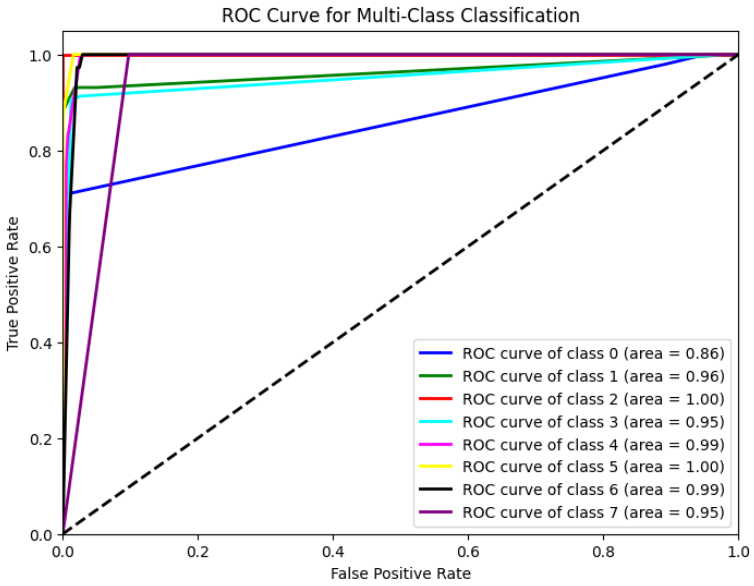


Figure 4.2.40: ROC Curve After Re-training

Figure 4.2.40 illustrates the ROC Curve for individual attack classes within the dataset after re-training RDS model, providing insights into their classification performance.

4.3 TEST CASES

```
from tensorflow.keras.models import load_model
# Load the trained model
model = load_model('RDS_model.h5')

# Define the input data (replace this with your actual input data)
input_data = np.array([[7,0]])

# Make predictions
predictions = model.predict(input_data)

# Print the predicted class
predicted_class = np.argmax(predictions)
print("Predicted Class:", predicted_class)
```

```
1/1 [==============================] - 0s 98ms/step
Predicted Class: 0
```

Figure 4.3.1: Test Case – 1

Figure 4.3.1 displays the test case-1 evaluated against the proposed model.

```
from tensorflow.keras.models import load_model
# Load the trained model
model = load_model('RDS_model.h5')

# Define the input data (replace this with your actual input data)
input_data = np.array([[7,4]])

# Make predictions
predictions = model.predict(input_data)

# Print the predicted class
predicted_class = np.argmax(predictions)
print("Predicted Class:", predicted_class)
```

```
1/1 [==============================] - 0s 257ms/step
Predicted Class: 4
```

Figure 4.3.2: Test Case – 2

Figure 4.3.2 displays the test case-2 evaluated against the proposed model.

Table 4.3.3: Test Cases

Case id	Test Case Description	Expected Result	Actual Result	Status
1	Provide input 0-7, input sample-1	0(Dos attack)	0(DoS attack)	Pass
2	Provide input 0-7, input sample-2	4(Scan)	4(Scan)	Pass
3	Provide input that is out of bounds	Error	Error: Input data is out of bounds	Pass

4.4 PERFORMANCE METRICS

Accuracy:

It represents the ratio of correctly predicted instances to total number of instances as shown in equation 1.
Formula: Accuracy = (TP+TN)/ (TP+FP+TN+FN) (1)

Confusion matrix:

Provides insights into how well the model correctly classifies each class and where it makes mistakes.

Recall:

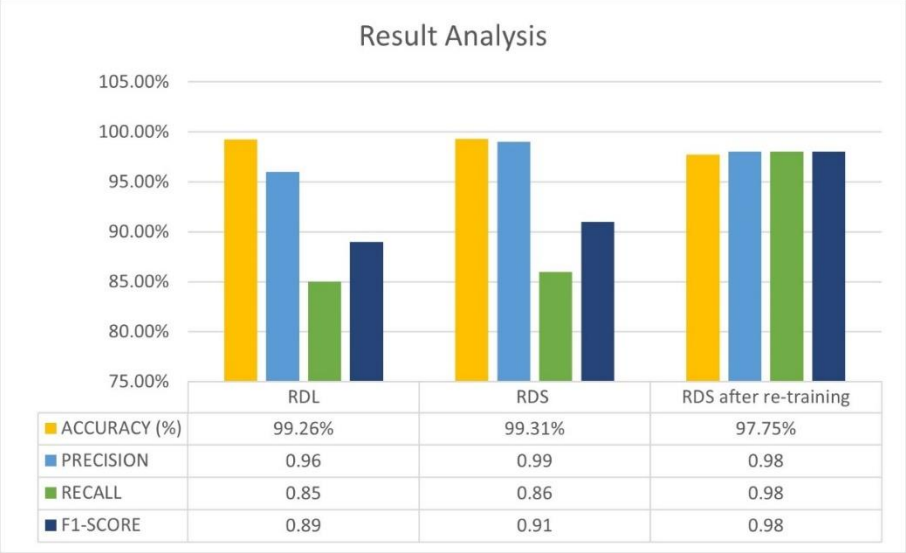
Evaluates the ability of the model to capture all the actual positive instances in the data as shown in equation 2.
Formula: Recall =TP/ (TP+FN) (2)

Precision:

It measures the accuracy of positive predictions as shown in equation 3.
Formula: Precision = TP/ (TP+FP) (3)

F1 score:

It is the harmonic mean of the precision and recall and provides a balance between the two as shown in equation 4.
Formula: F1 score = 2*(precision*recall) / (precision+recall) (4)

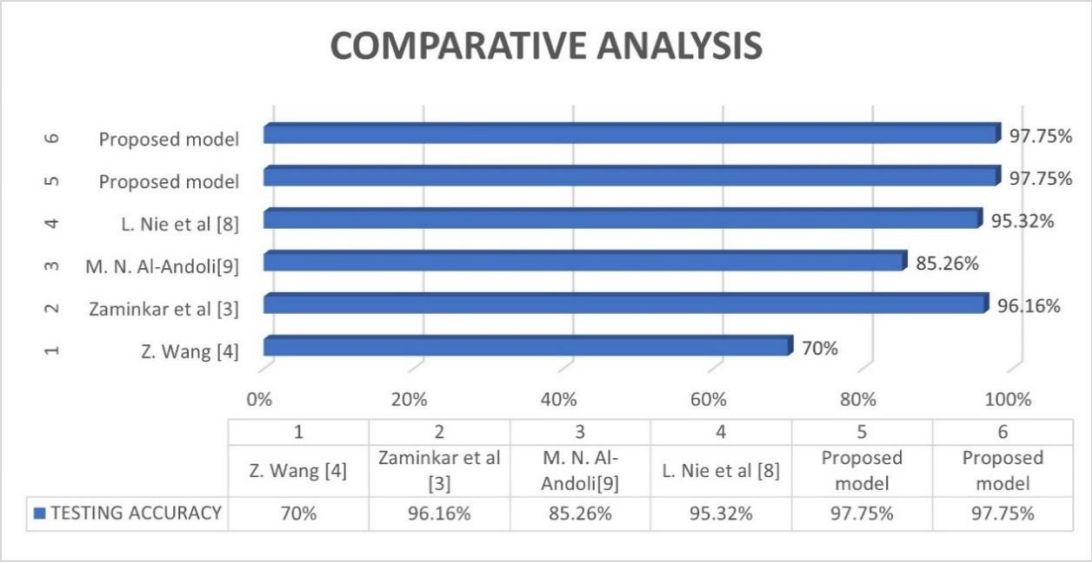


	RDL	RDS	RDS after re-training
ACCURACY (%)	99.26%	99.31%	97.75%
PRECISION	0.96	0.99	0.98
RECALL	0.85	0.86	0.98
F1-SCORE	0.89	0.91	0.98

Figure 4.4.1: Result Analysis

Figure 4.4.1 presents a bar graph illustrating the outcomes derived from RDS, RDL, and RDS after retraining.

4.5 COMPARATIVE ANALYSIS



	1	2	3	4	5	6
	Z. Wang [4]	Zaminkar et al [3]	M. N. Al-Andoli[9]	L. Nie et al [8]	Proposed model	Proposed model
TESTING ACCURACY	70%	96.16%	85.26%	95.32%	97.75%	97.75%

Figure 4.5.1: Comparative Analysis

In Figure 4.5.1, the accuracy of the compared papers is depicted alongside that of the proposed system.

CHAPTER 5

5.1 CONCLUSION

In conclusion, our project has made significant progress in enhancing the security of IoT-based smart cities against adversarial threats. Through the implementation of a hybrid ensemble model and adversarial training techniques, we have achieved promising results. Our experiments, conducted on the DS2OS dataset, demonstrated the effectiveness of our approach. Initially, our model exhibited a remarkable accuracy of 99% in threat detection. Even when subjected to adversarial data, it maintained a robust accuracy rate of 98%. These results highlights how the approach proposed in this paper make smart cities more secure against adversarial attacks.

5.2 FUTURE WORK

There is always a scope for enhancement in any developed system, especially when the project build using latest trending technology and has a good scope in future.

- Instead of detecting an intruder, detection systems can focus on identifying suspicious event and let the system administrator decide whether to start an investigation.
- development and implementation of adaptive threat modeling techniques within smart city environments. Adaptive threat modeling involves continuously analyzing emerging cyber threats, vulnerabilities, and potential system weaknesses to proactively identify and mitigate risks before they can

be exploited by attackers. By integrating adaptive threat modeling into cybersecurity strategies, smart cities can enhance their resilience against evolving cyber threats and ensure the ongoing security and reliability of critical infrastructure.

- Can explore techniques for conducting adversarial training while preserving the privacy of sensitive data in IoT environment.

REFERENCES

1. E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos and P. Burnap, "A Supervised Intrusion Detection System for Smart Home IoT Devices," in IEEE Internet of Things Journal, vol. 6, no. 5, pp. 9042-9053, Oct. 2019, doi: 10.1109/JIOT.2019.2926365.

2. Boukerche, Azzedine, and Rodolfo WL Coutinho. "Design guidelines for machine learning-based cybersecurity in internet of things." *IEEE Network* 35.1 (2020): 393-399.

3. Miyato, Takeru, et al. "Virtual adversarial training: a regularization method for supervised and semi-supervised learning." *IEEE transactions on pattern analysis and machine intelligence* 41.8 (2018): 1979-1993.

4. Z. Wang, "Deep_Learning-Based Intrusion Detection with Adversaries," in IEEE Access, vol. 6, pp. 38367-38384, 2018, doi: 10.1109/ACCESS.2018.2854599.

5. O. Ibitoye, O. Shafiq and A. Matrawy, "Analyzing Adversarial Attacks against Deep Learning for Intrusion Detection in IoT Networks," 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9014337.

6. R. Kumar Singh Gautam and E. A. Doegar, "An Ensemble Approach for Intrusion Detection System Using Machine Learning Algorithms," 2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India, 2018, pp. 14-15, doi: 10.1109/CONFLUENCE.2018.8442693.

7. N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," 2016 IEEE European Symposium on Security and Privacy (EuroS&P), Saarbruecken, Germany, 2016,pp. 372-387, doi: 10.1109/EuroSP.2016.36.

8. L. Nie et al., "Intrusion Detection for Secure Social Internet of Things Based on Collaborative Edge Computing: A Generative Adversarial Network-Based Approach," in IEEE Transactions on Computational Social Systems, vol. 9, no. 1, pp. 134-145, Feb. 2022, doi: 10.1109/TCSS.2021.3063538.

9. M. N. Al-Andoli, S. C. Tan, K. S. Sim, P. Y. Goh and C. P. Lim, "A Framework for Robust Deep Learning Models Against Adversarial Attacks Based on a Protection Layer Approach," in IEEE Access, vol. 12, pp. 17522-17540, 2024, doi: 10.1109/ACCESS.2024.3354699.

10. Hongling Jiang, Jinzhi Lin, Haiyan Kang, “FGMD: A robust detector against adversarial attacks in the IoT network”, Future Generation Computer Systems, Volume 132,2022, Pages 194-210, ISSN 0167-739X, https://doi.org/10.1016/j.future.2022.02.019.

11. M. Liu, Z. Zhang, Y. Chen, J. Ge and N. Zhao, "Adversarial Attack and Defense on Deep Learning for Air Transportation Communication Jamming," in IEEE Transactions on Intelligent Transportation Systems, vol. 25, no. 1, pp. 973-986, Jan. 2024, doi: 10.1109/TITS.2023.3262347

12. Mani, S., Sundan, B., Thangasamy, A., Govindaraj, L. (2022). A New Intrusion Detection and Prevention System Using a Hybrid Deep Neural Network in Cloud Environment. In: Pandian, A.P., Fernando, X., Haoxiang, W. (eds) Computer Networks, Big Data and IoT. Lecture Notes on Data Engineering and Communications Technologies, vol 117. Springer, Singapore. https://doi.org/10.1007/978-981-19-0898-9_73

13. N. Maheswaran, S. Bose, S. Sonny, M. Araventh, G. Tharun and R. J, "Effective Intrusion Detection System using Hybrid Ensemble Method for Cloud Computing," 2023 Second International Conference on Advances in Computational Intelligence and Communication (ICACIC), Puducherry, India, 2023, pp. 1-5, doi:10.1109/ICACIC59454.2023.10435091.

14. N. Maheswaran, G. Logeswari, S. Bose and T. Anitha, "A critical review on intrusion detection systems in IoT based on ML approach:A Survey," 2023

2nd International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN), Villupuram, India, 2023, pp. 1-8, doi: 10.1109/ICSTSN57873.2023.10151604.

15. B. Natarajan, S. Bose, N. Maheswaran, G. Logeswari and T. Anitha, "A Survey: An Effective Utilization of Machine Learning Algorithms in IoT Based Intrusion Detection System," 2023 12th International Conference on Advanced Computing (ICoAC), Chennai, India, 2023, pp. 1-7, doi: 10.1109/ICoAC59537.2023.10249672.

16. T. -T. -H. Le, R. W. Wardhani, D. S. C. Putranto, U. Jo and H. Kim, "Toward Enhanced Attack Detection and Explanation in Intrusion Detection System-Based IoT Environment Data," in IEEE Access, vol. 11, pp. 131661-131676, 2023, doi: 10.1109/ACCESS.2023.3336678.

17. Maheswaran, N., Bose, S., Logeswari, G., Anitha, T. (2023). Hybrid Intrusion Detection System Using Machine Learning Algorithm. In: Khanna, A., Polkowski, Z., Castillo, O. (eds) Proceedings of Data Analytics and Management . Lecture Notes in Networks and Systems, vol 572. Springer, Singapore. https://doi.org/10.1007/978-981-19-7615-5_30

18. B. Natarajan, S. Bose, N. Maheswaran, G. Logeswari and T. Anitha, "A New High-Performance Feature Selection Method for Machine Learning-Based IOT Intrusion Detection," 2023 12th International Conference on Advanced Computing (ICoAC), Chennai, India, 2023, pp. 1-8, doi: 10.1109/ICoAC59537.2023.10249916.

19. E. Gelenbe and M. Nakip, "IoT Network Cybersecurity Assessment With the Associated Random Neural Network," in IEEE Access, vol. 11, pp. 85501-85512, 2023, doi: 10.1109/ACCESS.2023.3297977.

20. H. Zahan, M. W. Al Azad, I. Ali and S. Mastorakis, "IoT-AD: A Framework to Detect Anomalies Among Interconnected IoT Devices," in IEEE Internet of Things Journal, vol. 11, no. 1, pp. 478-489, 1 Jan.1, 2024, doi: 10.1109/JIOT.2023.3285714.