



Addis Ababa University

Addis Ababa Institute of Technology

School of Information Technology and Engineering

MASTERS THESIS

**Lightweight Intrusion Detection System for IoT with Improved
Feature Engineering and Advanced Dynamic Quantization**

Author:

Semachew Fasika

Advisor:

Henock Mulugeta (Dr.)

*A thesis submitted in fulfillment of the requirements
for the degree of MSc in*

Cyber Security

November 14, 2024

Approval Sheet



This certification affirms that the thesis authored by Semachew Fasika, titled Lightweight Intrusion Detection System for IoT with Improved Feature Engineering and Advanced Dynamic Quantization has been submitted to satisfy the partial requirements for the Master of Sciences degree in Cyber Security. The document meets to university regulations and attains accepted standards in terms of originality and quality.

Approved by Board of Examiners

Name	Signature	Date
Dr. Elefelious Getachew		
_____	_____	_____
(School dean)		
Dr. Henock Mulugeta		
_____	_____	_____
(Advisor)		
Dr. Fitsum Assamnew		
_____	_____	_____
(External Examiner)		
Dr. Sileshi Demesie		
_____	_____	_____
(Internal Examiner)		

© 2024 by Semachew Fasika

All rights reserved. No part of this research may be produced or broadcast in any form or by any means mechanical, electronic, including photocopying, recording or any information storage without first written permission from the author.

Acknowledgements

I express my acknowledgment to the Divine for the abundant blessings that have illuminated my path throughout my research endeavor. I am profoundly appreciative of the celestial grace that has steered me through difficulties, empowering me to surmount obstacles and achieve this noteworthy accomplishment.

I am delighted to express my gratitude to the remarkable individuals who played pivotal roles in the success of this undertaking. My heartfelt thanks go to my advisor, Dr. Henock Mulugeta, for his unwavering support, invaluable comments, and continuous encouragement throughout my research. His guidance, patience, and optimistic spirit have been vital in shaping the outcome of my work. I am profoundly grateful for the immense knowledge and expertise he shared, making him an exceptional mentor for my MSc study.

Finally, I would like to thank Addis Ababa University and the Information Network Security Administration for their support and resources. I want to thank Abubakar Yesuf, Asmamaw Mersha, Mubarek Mohammed and Zeru Kifle for their continuous support and resources, which have been indispensable in this research. This acknowledgment is an evidence to the collaborative effort that made this achievement possible.

Declaration of Authorship

I, Semachew Fasika, declare that this thesis titled, “Lightweight Intrusion Detection System for IoT with Improved Feature Engineering and Advanced Dynamic Quantization” and the work presented in it are my own. I confirm that:

- This research was conducted primarily during my pursuit of a master’s degree in Cyber Security at Addis Ababa University.
- No portion of the thesis has been previously submitted for any degree or qualification at Addis Ababa University or another academic institution.
- Proper attribution is consistently provided for all referenced published works.
- All quoted material from external sources is accompanied by appropriate citations. Apart from these quotations, the content of this thesis is original and solely my own work.
- I have duly acknowledged all significant sources of assistance received during the course of this research.

Name

Signature

Date

Semachew Fasika

(Name of Student)

This thesis has been submitted for examination with my approval as a university advisor.

Dr. Henock Mulugeta

(Name of Advisor)

Abstract

In recent years, the proliferation of Internet of Things (IoT) devices and applications has experienced a significant surge globally, owing to their inherent advantages in enhancing both business and industrial landscapes, as well as facilitating improvements in individuals' daily routines. Nevertheless, IoT devices are not immune to malicious attacks, which results potential negative consequences and malfunctioning of IoT devices, therefore, attack detection and classification becomes an important issue in IoT devices. This research proposes a lightweight hybrid deep learning model (DNN-BiLSTM) to detect and classify attacks in an IoT system with improved feature engineering and advanced quantization.

Although leveraging hybrid deep learning model which combines DNN alongside BiLSTM, facilitates the extraction of intricate network features in a nonlinear and bidirectional manner, aiding in the identification of complex attack patterns and behaviors, its implementation on IoT devices remains challenging. To mitigate the constraints inherent in IoT devices, this research incorporates improved feature engineering, specifically Redundancy-Adjusted Logistic Mutual Information Feature Selection (RAL-MIFS) combined with a two-stage IPCA algorithm. Additionally, advanced quantization (QAT + PTDQ) techniques, alongside advanced Optuna for hyperparameter optimization, are utilized to enhance computational efficiency without compromising detection accuracy.

Experimental evaluations were conducted on the CIC IDS2017 and CICIOT2023 datasets to assess the performance of a quantized DNN-BiLSTMQ model. The model demonstrated superior detection accuracy & computational efficiency compared to state-of-the-art methods. On the CIC IDS2017 dataset, it achieved a detection accuracy of 99.73% with a model size of 25.6 KB, while on the CICIOT2023 dataset, it achieved a detection accuracy of 93.95% with a model size of 31.3 KB. These results highlight the potential of the quantized DNN-BiLSTMQ model for efficient and accurate cyber attack detection on IoT.

Keywords: Attack detection, Cybersecurity, Cyber attack, Intrusion, Lightweight deep learning, Feature engineering, IoT, DNN, BiLSTM, Advanced Dynamic Quantization and Optimization, RAL-MIFS

Contents

Approval Sheet	i
Acknowledgements	iii
Declaration of Authorship	iv
Abstract	v
List of Figures	ix
List of Tables	x
List of Abbreviations	xi
1 Introduction	1
1.1 The Overview	1
1.2 Motivation of the Research	4
1.3 Problem Statement	5
1.4 Research Questions	6
1.5 Research Objective	7
1.5.1 General Objective	7
1.5.2 Specific Objective	7
1.6 Expected Contribution of the Research	8
1.7 Scope of the Research	9
1.7.1 In Scope of the Research	9
1.7.2 Delimitation of the Research	9
1.8 Organization of Research	10
2 Related Work	11
2.1 Cyber Attack Detection for IoT	11
2.2 Improved Feature engineering for Cyber Attack Detection on IoT	14
2.3 Lightweight Cyber Attack Detection Model	16

2.4	Summary and Research Gap	20
3	Methodology	21
3.1	Research Methodology	21
3.2	Dataset and Preprocessing	21
3.2.1	Dataset Overview	21
3.2.2	Data Preprocessing	24
3.2.2.1	Dataset Cleaning	25
3.2.2.2	Label Encoding	25
3.2.2.3	Feature Engineering	26
3.2.2.4	Normalization	27
3.2.2.5	Data Splitting	27
3.3	Proposed Lightweighting Architecture	28
3.4	Improved Feature Engineering	31
3.4.1	Redundancy-Adjusted Logistic MIFS	31
3.4.2	Two stage IPCA	33
3.4.3	RAL-MISF + Two Stage IPCA Pseudocode	35
3.5	Hybrid DNN-BiLSTM Model	37
3.6	Model Lightweighting	42
3.6.1	Lightweighting Techniques	42
3.6.2	Advanced Dynamic Quantization	43
3.6.3	Quantized DNN-BiLSTMQ Model	46
3.7	Advanced Model Optimization	48
3.7.1	Advantages of Optuna	48
3.7.2	Advanced Optuna Implementation Strategy	49
4	Experimental Results and Discussion	52
4.1	Experimental Setup and Evaluation Metrics	52
4.1.1	Experimental Setup	52
4.1.2	Evaluation Metrics	53
4.2	Analyzing of Performance Measures on the Dataset	55
4.2.1	Comparsion of IPCA with Two stage IPCA	55
4.2.2	Performance Measures on CIC IDS2017 Dataset	57
4.2.3	Performance Measure on CICIOT2023 dataset	61
4.3	Analysis of Lightweight Model Performance	65
4.3.1	Model Lightweightness Analysis on CIC IDS2017 Dataset	65
4.3.2	Model Lightweightness Analysis on CICIOT2023 Dataset	66
4.3.3	Key Factors for Lightweight Attributes	66

4.4	Comparing Proposed Architecture with State-of-the-Art Techniques . . .	67
4.5	Deployment Architecture	68
5	Conclusion	70
	Bibliography	71
A	Appendix	77
A.1	Glossary of All Terms	77
B	Appendix	80
B.1	Source Code	80
B.1.1	Data preprocessing	80
B.1.2	Optuna parameter optimization with RLA-MIFS	85
B.1.3	Models	92
B.1.3.1	BiLSTM	92
B.1.3.2	CNN	93
B.1.3.3	DNN	95
B.1.3.4	DNN-BiLSTMQ	96

List of Figures

1.1	Cyber Attacks and Targeted Layers in IoT	2
3.1	Proposed lightweight hybrid model	29
3.2	DNN_BiLSTM Model Structure	38
3.3	LSTM Structure	39
3.4	DNNBiLSTMQ Model Structure	47
4.1	Comparsion of RAL-MIFS + IPCA/RAL-MIFS + Two stage IPCA	55
4.2	BiLSTM Quantitative Assessment	59
4.3	DNN Quantitative Assessment	59
4.4	CNN Quantitative Assessment	60
4.5	DNN-BiLSTMQ Quantitative Assessment	60
4.6	BiLSTM Quantitative Assessment	62
4.7	CNN Quantitative Assessment	62
4.8	DNN Quantitative Assessment	63
4.9	DNN-BiLSTMQ Quantitative Assessment	63
4.10	Proposed implementation architecture	69

List of Tables

2.1	Summary of surveyed associated research publication of machine learning based anomaly ADS.	13
2.2	Summary of surveyed associated research publication of Lightweight Intrusion detection systems	19
3.1	CIC IDS2017 dataset attack distribution.	22
3.2	CICIoT2023 dataset attack distribution.	23
3.3	Selected features CIC-IDS2017	32
3.4	Selected features CICIoT2023	32
4.1	Comparison of multiclassification performance on CIC IDS2017 dataset.	57
4.2	Comparison of multiclassification performance on CICIoT2023 dataset.	61
4.3	Comparison of Attack Detection Accuracy on Different Datasets	65
4.4	Comparison of models' multiclassification effectiveness.	67
4.5	Comparison of models' lightweightness.	67

List of Abbreviations

ACCS	Australian Centre for Cyber Security
ANN	Artificial Neural Network
BiLSTM	Bidirectional Long Short-Term Memory
CNN	Convolutional Neural Network
CIA	Confidentiality Integrity and Availability
CPS	Cyber-Physical System
DBN	Deep Belief Network
DL	Deep Learning
DL-ACS	Deep Learning based Attack Classification System
DL-ADS	Deep Learning based Attack Detection System
DNN	Deep Neural Network
DT	Decision Tree
DOS	Denial of Service
DDOS	Distributed Denial of Service
DNN-BiLSTM	Bidirectional Long Short-Term Memory with Deep Neural Network
ETC	xtra tree classifier
FFN	Feed Forward Neural Network
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
GANs	Generative Adversarial Networks
ICA	Independent Component Analysis
IDS	Intrusion Detection Systems
IGAN	Imbalanced Generative Adversarial Network

HDL-ADS	Hybrid Deep Learning Based Attack Detection System
IIoT	Industrial IoT
INSA	Information Network Security Administration
IoT	Internet of Things
PCA	Principal Component Analysis
IPCA	Incremental Principal Component Analysis
LSTM	Long Short Term Memory
MIFS	Mutual information feature selection
MITM	Man-In-The-Middle
ML	Machine Learning
NIDS	Network Intrusion Detection Systems
PCA	Principal Component Analysis
PTDQ	Post-training Dynamic Quantization
QAT	Quantization Aware Training
ReLU	Rectified Linear Unit
RF	Random Forest
RFE	Recursive Feature Elimination
RAL-MIFS	Redundancy-Adjusted Logistic MIFS
RNN	Recurrent Neural Network
SCADA	Supervisory Control and Data Acquisition
TCP	Transfer Control Protocol
TP	True Positive
UDP	User Datagram Protocol

Chapter 1

Introduction

1.1 The Overview

Internet services are broadly used in many businesses, industries, education, health, transport, agriculture and many aspects of human life. In the previous 20 years, advances in information technology have made network infrastructure an integral part of individuals and organizations (Kiourtis et al., 2018).

The new paradigm called Internet of Things (IoT) focused on connecting large numbers of smart physical devices to modernize and integrate diverse domains, thereby improving quality of life. "Internet of Things" created originally by Kevin Ashton in 1999 (Tian, Li, and Liu, 2019).

Our capacity to understand the ecosystem and all the globe has significantly increased because of very sophisticated cyber attack and smart IoT. CPS is regularly used when we speak about a large, and interconnected devices, for example smart Grid, Smart Retail, Smart Supply Chain, Smart Agriculture, Smart Industry, Smart Transportation, Smart Health, Smart Wearable, Smart Housing & Buildings and Smart City. In the other hand, IoT is usually used to refer also small, and interconnected devices, such as systems used in smart home (Gopali and Siami Namin, 2022). IoT technology has affected almost every individual's everyday life with its vast and vital applications.

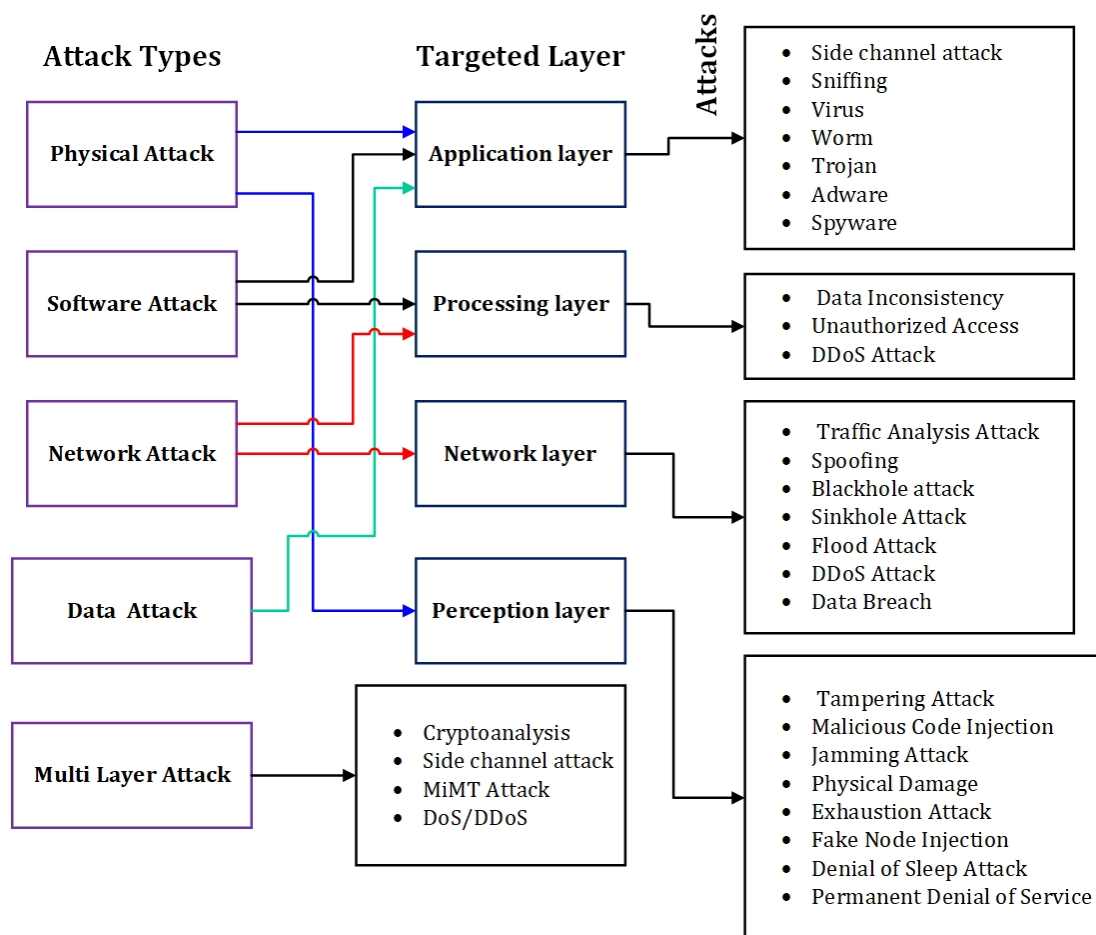
In 2020, it was predicted that global spending on IoT technology would reach 749 billion dollars (Gopali and Siami Namin, 2022). IoT systems continue to proliferate, connected and distributed across a broad range of services & applications. Utilization of IoT devices all over the globe is expected to pass 125 billion in 2030 (Otoum, Liu, and Nayak, 2022).

As the deployment of IoT-based technologies increases, the security of these connected devices also becomes paramount to ensure their successful operation. According to

data presented by (Gopali and Siami Namin, 2022), the period from 2020 to 2021 saw a huge number of malicious attacks globally through Internet of Things (IoT) instruments, with October 2020 alone witnessing 10.83 million incidents. These malicious activities have the ability to compromise systems and result in data breaches.

IoT is a promising and insightful technology with amazing results and potential for adoption and growth, even though IoT infrastructure is exposed to different types of cyber-attacks and incidents (Shi and Dustdar, 2016). This is by the limited processing power, storage, storage capacity, and communication capacity of small energy-conscious endpoint devices present in IoT infrastructure. In fact, confidentiality, integrity, & availability (CIA) is one of the major challenges in the ecosystem of IoT (Shi and Dustdar, 2016). Figure 1.1 shows various cyber attacks against IoT systems.

FIGURE 1.1: Cyber Attacks and Targeted Layers in IoT (Alshahrani, 2021)



Signature-based attack detection occupies a significant position within the domain of cybersecurity. This approach relies on pre-established patterns-based signatures indicative of known threats to discern and counteract malicious activities. It can give us

highly accurate cyber attack detection performance for cyber attacks that correlates pre-stored attack patterns (network traffic patterns, system call sequences, etc.) (Alsulami et al., 2022), the problem will occur and even drastically increase when we discover a new (zero-day) attack. This is because traditional signature-based cyber attack recognition efficiency relies on the pre-knowledge of different cyber attack signatures. Hence, they can identify cyber attack only if it is pre-deposited in the database.

Hence, to address this bottleneck, anomaly-based cyber attack detection has been suggested to change the traditional cyber attack detection by employing smarter techniques. Anomaly-based attack detection defines a profile that represents "normal" behavior and after that detects the variation, instead of matching cyber attack signatures to existing cyber attack patterns. This allows you to detect zero day cyber attacks. However, in a sophisticated and highly distributed like the IoT ecosystem, it still doesn't accurately detect all zero day cyber attacks, and the false alert rate is still costly. As such, many zero-day attacks go undetected due to the IoT devices' limitations which exist and traditional advanced detection methods which are not able to learn and are incapable of detecting unknown attacks. Such functions are usually facilitated by vital and essential defenses (Saba et al., 2022).

Deep learning is a specialized form of machine learning within the domain of artificial intelligence (AI) that applies deep artificial neural networks, also known as deep neural networks. They are machine learning techniques that simulate the learning process of the human brain. The human brain is made up of cells called neurons in a neural network. Similarly, in human brain, all cells are connected by axons and dendrites in connection areas known as synapses. These connections contain weights that behave like the connections between neurons in human brain when found in an ANN (Alkhatani and Aldhyani, 2021).

The main difference between traditional neural networks & deep neural networks is the quantity of layers. Deep neural networks use several hidden layers to provide the data a high degree of abstraction. You can learn the properties of data. This process of feature engineering is performed using a huge number of features fed to algorithms by deep learning to produce results with a form of best feature followed by classification, identification or data generation. The main motivation for using deep learning in various fields has been to organize and analyze bulk amounts of data. As a consequence of the swift advancements observed in Internet-connected systems and Artificial Intelligence (AI) over the past years, AI was applicable, encompassing DL and ML, has become prevalent across various domains, notably in cybersecurity applications such as cyber attack detection, malware detection, and spam filtering (Zhang et al., 2022).

1.2 Motivation of the Research

The swift usage of Internet of Things (IoT) technology has led to their widespread connectivity across diverse applications and services. Despite the promising potential of IoT technology, its expansion also exposes vulnerabilities to various cyber threats and attacks. These vulnerabilities arise from limitations in processing power, storage, memory, and communication capabilities inherent in the small, energy-efficient endpoint devices comprising IoT infrastructures. Satisfying confidentiality, integrity, & availability (CIA) within the IoT ecosystem presents significant challenges.

While conventional cyber attack detection methods, such as signature-based approaches, excel at accurately identifying known attacks by matching them with pre-stored intrusion patterns (e.g., sequences of system calls, network traffic patterns) (Alsulami et al., 2022), they stumble when confronted with novel attacks, including zero-day threats. This limitation stems from their reliance on prior knowledge of attack signatures, rendering them ineffective against previously unseen threats. Despite the efficacy of machine learning methodology in detecting new cyber threats, they often suffer from high false positive rates.

To overcome this challenge, there is high demand to transition from conventional cyber attack detection methods to more intelligent and resource-efficient techniques. Anomaly-based cyber attack detection, for example, establishes a standard of "normal" behavior and identifies change from the baseline. However, it struggles to accurately detect all unknown attacks in dynamic environments like IoT ecosystems, resulting in a persistently high false detection rate. Consequently, many zero-day attacks evade detection because of the resource-constrainedness of IoT devices & conventional cyber attack detection methods.

Addressing this issue involves not only enhancing the intelligence of detection mechanisms but also optimizing them to be used by resource-constrained IoT devices. Developing lightweight algorithms tailored to the limited memory and power of IoT devices is crucial for ensuring effective cyber attack detection without compromising system performance. The primary objective remains the creation of an intelligent system capable of detecting and identifying cyber attacks in a cost-effective manner, in real-time, thereby enabling prompt mitigation of their impacts.

1.3 Problem Statement

The global adoption of smart sensors has accelerated due to their effectiveness in enhancing business operations, industrial workflows, and daily routines. Smart technologies now support diverse sectors, such as healthcare, smart homes, smart cities, and smart grids (Din et al., 2019). Despite their broad application, IoT devices remain vulnerable to security threats, posing risks of disruption and compromised functionality across interconnected systems due to their resource constraints.

The growing data capacity and connectivity of IoT devices increase their vulnerability to cyber threats, necessitating robust detection and classification systems. Conventional machine learning models often struggle with high false positive rates (Tian-yi, Shi-yue, and Jun-yi, 2019) and limited adaptability to emerging attack patterns. While deep learning models effectively capture complex patterns in network traffic, their deployment in IoT environments faces challenges due to cost, scalability, and resource constraints (Agbedanu et al., 2022; Ullah et al., 2022; Zhao et al., 2021a; Wang et al., 2023; Murugesan and Maheswaravenkatesh, 2024). Additionally, large feature sets can increase model complexity, further straining limited IoT resources. Ensuring model generalizability across varied IoT environments is also difficult, as models trained on specific datasets may fail to adapt effectively to diverse real-world scenarios (Wang et al., 2023).

To address these challenges, this research proposes a resource-efficient hybrid deep learning model specifically designed for IoT security. The model leverages advanced quantization techniques, Quantization-Aware Training (QAT) and Post-Training Dynamic Quantization (PTDQ), to enhance performance in resource constrained environments without compromising detection accuracy. Additionally, it incorporates robust feature engineering methods, including Redendency Adgusted Logistic Regression-Mutual Information Feature Selection (RAL-MIFS) and a two-stage Incremental Principal Component Analysis (IPCA), to reduce dataset dimensionality by extracting the most relevant features. This approach streamlines the model, enabling it to effectively detect subtle attack patterns in IoT data streams. By minimizing model size and reducing false positives, this research strengthens the resilience and reliability of IoT networks against emerging cyber threats, ensuring efficient and precise attack detection and categorization.

1.4 Research Questions

RQ1: What is the impact of advanced feature engineering techniques, such as RAL-MIFS and two-stage IPCA, on the accuracy and efficiency of cyber attack detection and classification within IoT networks?

RQ2: How can advanced quantization technique, including QAT and PTDQ, be applied to optimize and reduce the complexity of deep learning algorithms for real-time cyber attack detection and classification in resource-constrained IoT environments?

1.5 Research Objective

1.5.1 General Objective

The main objective of the study is to develop a lightweight predictive model to cyber attack identification and categorization in an IoT system with improved feature engineering and advanced dynamic quantization.

1.5.2 Specific Objective

The research was guided by the following specific objectives:

- To evaluate and select effective cyber attack detection and classification algorithms suited for IoT systems, focusing on accuracy, efficiency, and adaptability.
- To extract critical features from the dataset to yield a very light weight model without compromising accuracy.
- To develop a lightweight model without compromising accuracy for resource-limited IoT environments.
- To achieve high accuracy, low resource consumption, and minimized false positive rates across diverse attack types.
- To perform a comprehensive analysis of experimental results, examining model accuracy, resource efficiency, and false positive rates across different attack types.

1.6 Expected Contribution of the Research

This study propose a cyber attack identification and categorization system capable of identifying and categorizing zero-day attacks of typical malicious IoT traffic employing lightweight hybrid deep learning model. Deploying attack detection systems intended for traditional computers in IoT environments is practically infeasible due to the computational, memory, power, & CPU constraints inherent in these devices. In this regard, this research come up with a lightweight attack identification and categorization for IoT devices. The main expected research's contributions can be summed up as follows:

- A comprehensive and robust approach, Redundancy-Adjusted Logistic MIFS (RAL-MIFS) + two stage IPCA feature engineering and data preparation system, that greatly enhances system performance. Standard deep learning model performance measures, including accuracy, confusion matrix, precision, recall, F-score, memory, running time, model size, and FLOPS, were used to evaluate the system.
- We present a comprehensive and lightweight cyber attack identification and categorization capable of recognizing and categorizing IoT attacks of a dataset into normal and to different attacks types.
- We present an advanced quantization and advanced optuna hyper parameter optimization method that greatly reduce model size and make it suitable for resource constrained IoT devices.
- We examine our discoveries with other related state-of-the-art models, deep-learning based attack detection systems (DL-ADSs), and attack classification systems (DL-ACSs) by testing with identical dataset.

1.7 Scope of the Research

1.7.1 In Scope of the Research

- To develop a feature engineering system using RAL-MIFS combined with a two-stage IPCA approach.
- To implement advanced quantization techniques and optuna-based hyperparameter optimization.
- To detect and classify cyber attacks on IoT systems using a lightweight deep-learning blended model.
- To compare findings with other state-of-the-art techniques.

1.7.2 Delimitation of the Research

- Real-time implementation of the research

1.8 Organization of Research

The structure of this thesis is in the subsequent order:

Chapter 1 – Introduction: provides an introduction to IoT and introduces neurons and neural networks. Also gives an overview of the attack detection, feature engineering and lightweighting models. This chapter also establishes the problem statement, research questions, research goals and research contributions along with the research delimitation.

Chapter 2 – Related Work: discusses Internet-of-Things (IoT) along with IoT architecture and security concerns and the sectors where IoT systems are applied. This chapter also establishes the background in Deep Learning (DL), and Machine Learning (ML) techniques and their implementation in IoT. This chapter also presents the literature review of similar works done by other researchers, existing methods and their issues and identify a potential research gap.

Chapter 3 – Methodology discusses about the research method, dataset , preprocessing the proposed architecture for improved feature engineering, an architecture for lightweight cyber attack detection and classification system with details, model and also an advanced dynamic quantization system with advanced optuna hyper parameter optimization.

Chapter 4 – Experimental Result and Discussion: provides the implementation of the aforementioned proposed model through PyTorch framework and Python programming language. Also the simulation outcomes and the performance evaluation of the developed model with various parameters. Comparison of our model with other and also present real time deployment architecture.

Chapter 5 – Conclusions and Future Work: presents the conclusion drawn out of the project including the possible future works that might be accomplished to enhance and upgrade the research.

Chapter 2

Related Work

2.1 Cyber Attack Detection for IoT

The proliferation of computers, smartphones, and other internet-enabled devices has made the world vulnerable to cyber attacks. confidentiality and safety challenges are also evolving and increasing as the IoT touches everyone every day (Sadiq et al., 2019; Almomani and Alromi, 2020). Additionally, the security threats exposed by IoT devices have varying impacts depending on the domain in which they operate. Each device can suffer from unique vulnerabilities specific to the characteristics, processes, and requirements of its domain, such as smart grids, industrial applications, and autonomous vehicles, smart home, healthcare, smart city.

Security measures consist of several methods aimed at securing, maintaining, and recovering system security from malicious attacks. These attacks can occur for a variety of reasons, including lack of adherence to security standards by manufacturers, lack of user knowledge and awareness, problems with update management processes, lack of physical hardening, and botnet attacks. Therefore, these issues with his IoT devices in a real environment increase the exposure to intrusion threats. These threats include, for example, black holes (interruption of network traffic) (Kumar et al., 2021), memory channel attacks (transfer of information by unauthorized processes) (Gabsi et al., 2021), buffer overflows (exceeding buffer limits) (Mullen and Meany, 2019), and SQL injection (Rajendran et al., 2019).

The continued rise in information leaks and attacks from IoT devices has led to an acceleration and increase in privacy and security research in the literature (Elhoseny et al., 2021; Rizi and Seno, 2022). Although some precautions have been taken, IoT systems and devices are still exposed to evil-intentioned hackers and actors. Therefore, the need for early detection of intrusive activity to mitigate the negative impact of these devices is more urgent than ever. Fortunately, such threats can be mitigated by a detection

measure called attack detection. As a result, standard intrusion detection techniques are being improved by incorporating machine learning techniques (Fatani et al., 2022; Qaddoura et al., 2021; Maheswaran et al., 2022; Mendonca et al., 2022).

(Fatani et al., 2022) made use of the perks of swarm intelligence (SI) techniques to present a novel feature engineering strategy for the attack recognition system. BoT-IoT, NSL-KDD, CIC2017, and KDD99 are well-known public datasets that were utilized to assess the efficacy of the suggested cyber attack detection method. For cyber attack detection, three sophisticated and popular DL models were proposed by (Alkahtani and Aldhyani, 2021). CNN, LSTM, and a CNN-LSTM hybrid model experiment were carried out by the authors. They assessed these DL models usage of the IoTID20 dataset. Conventional network attack detection techniques can be effectively enhanced with machine learning technology.

(Tian, Li, and Liu, 2019), selected the UNSW-NB15 dataset as their research subject. The proposed strategy which is superior to PCA-oriented approach and other deep learning methods, as confirmed by their mix of deep learning and shallow learning techniques. They used deep autoencoder strategy to lessen the features of the data and then supplied to SVM with the compressed data to determine the ideal classifier parameters SVM by means of artificial bee-colony algorithm. But as training progressed, this strategy also gradually showed its shortcomings.

As per (Khan, 2021), a novel hybrid attack detection system (HIDS) called Hybrid CRNN attack Detection System (HCRNNIDS) has been introduced, employing convolutional recurrent neural network. This system integrate the strengths of convolutional neural networks (CNNs) for extraction of local features & recurrent neural networks (RNNs) for capturing temporal features. By harnessing these techniques, the HCRNNIDS demonstrates significant advancements in cyber attack detection, achieving remarkable accuracy rates of 97.75% in detecting malicious cyber attacks on the CSE-CIC-IDS2018 dataset under 10-fold cross-validation. This innovative approach outperforms conventional cyber-attack detection methodologies, marking a substantial stride towards more effective cybersecurity measures in network environments.

TABLE 2.1: Summary of surveyed associated research publication of machine learning based anomaly ADS.

Ref.	Datasets	Algorithms	Results	Cyber-attack	Limitations
(Abdulhammed et al., 2019)	IoTID20	Particle Swarm (PSO) XG Boost, RF	98% 83%	Mirai, DoS, Scan, MITM	Low multiclass detection rate Algorithm not lightweigjht
(Fatani et al., 2022)	NSL-KDD IoTID20	Convolutional neural network Long short-term memory CNN-LSTM	96.6% 99.82% 98.8%	Norm, DoS, DoS, Scan, Scan, MITM	Low detection & classification Rate Big Model size
(Alsulami et al., 2022)	IoTID20	Shallow Neural Network, Decision Trees, Bagged Tree K-Nearest Neighbor Support Vector Machine	100% 100% 99.80% 99.40%	Mirai, DoS, Scan, MITM	The research discarded sub-category feature Algorithm is not lightweight
(Kaur, Pal, and Singh, 2018)	Malicia	Hybridization of K-Means & Firefly Algorithm	97.95% 99.80%	DoS, U2R, R2L & Probe	Detect the advanced malware by analyzing feature & grouping them.
(Yao et al., 2023)	WSN-DS UNSW-NB15 KDD CUP99	One-Class Bidirectional GRU Autoencoder	97.91% 98.20% 98.23%	Exploits, Fuzzers DoS, Shellcode Analysis, Backdoor	Low detection rate
(Boppana and Bagade, 2023)	IoTID20 N-BaIoT	Deep Neural Network	93% 92% 92% 92% 92%	Norm, DoS, Mirai, DoS, Scan, MITM Probe, R2L Normal, Bashlite,	Low detection accuracy & Algorithm lightweightness
(Agbedanu et al., 2022)	TON-IoT	Ensemble Learning (Gaussian Naive Bayes & Hoeffding tree)	99%	XSS, backdoor, Ransomware, Scanning, MiTM DDoS, DoS	CPU usage, memory, & energy consumption not evaluated

2.2 Improved Feature engineering for Cyber Attack Detection on IoT

Feature engineering is indispensable in the sphere of cyber threat detection within the Internet of Things (IoT) environments, as it aids in extracting relevant information from raw data, thereby enabling the development of simple and accurate detection systems. Numerous research endeavors have explored diverse methods of feature engineering tailored specifically for IoT cyber attack detection. For example, (Wang et al., 2022) investigated feature engineering techniques leveraging deep learning methodologies. Their study aimed to enhance the discriminatory power of features extracted from Internet of Thing network traffic data. By employing CNNs and RNNs, they were able to automatically extract intricate patterns from network traffic data. This approach significantly bolstered the accuracy of cyber attack identification in Internet of Things networks.

Data engineering and feature preprocessing therefore formulated the core part of the DL-based ADS model. This has an essential role in making the raw data collected from the IoT environment available for further analysis & prediction (Alsulami et al., 2022).

(Alalhareth and Hong, 2023) proposed improved feature selection method named Logistic Redundancy Coefficient Gradual Upweighting MIFS that assesses individual features instead of checking them with same characteristics of the already selected features. In contrast to other feature selection methods currently in use, logistic function was used to determine a feature's redundancy score.

(Wang et al., 2023) used incremental principal component analysis (IPCA) techniques for feature dimension reduction. IPCA learn incrementally data features & is more convenient for real-time monitoring of the network traffic within IoT environment.

A integrated wrapper-based feature-election algorithm based on the ideas of the Tabu Search (TS)-based expectation criteria and the Cellular Automata (CA) engine is proposed by (Nazir et al., 2023). To assess the fitness of the chosen features, they employed an ensemble learning classifier called Random Forest (RF). The TON_IoT data was used to test the suggested algorithm, CAT-S. Their results show that CAT-S performs exceptionally good with respect to feature reduction and attack detection accuracy. The algorithm surpass the other methods, which show high accuracy values, with an efficiency of 99.50%. Furthermore, CAT-S maintains outstanding classification performance with a significant reduction in features to 13 and achieves an exceptionally low false alarm ratio of 0.004, demonstrating its superiority in differentiating between

benign and malicious instances. Additionally, the result from the simulation show that CAT-S algorithm decrease feature numbers by over 70% while also significantly improving attack-detection efficiency and false alarm ratio (FAR).

A novel feature engineering strategy and DCNN-based model for identifying evil-intended attacks in Internet of Things networks are presented by (Ullah et al., 2022). Optimizing performance while lowering computational power was the goal. The suggested model successfully lowers computing power while increasing performance. For binary classification, they get 98.12% , and for multiclassification of 77.0% accuracy.

(Said, Sabir, and Askerzade, 2023) suggested a unique strategy to enhance network cyber attack detection by employing both binary and multi-label categorization techniques. They modeled hybrid model combining CNN and BiLSTM network architectures. Assessment of the effectiveness of model was conducted using widely recognized datasets such as UNSW-NB15 and NSL-KDD, along with the specialized InSDN dataset dedicated to Software-Defined Networking (SDN). Their findings demonstrate how well the suggested approach works to accomplish high accuracy rates while significantly reducing the training time. Notably, The model demonstrated an amazing level of precision of 98.42%, showcasing its enhanced performance relative to existing methods.

Additionally, recent advancements in this field include the work of (Gao et al., 2023), proposed a unique feature engineering strategy based on the algorithm GNNs for IoT cyber attack detection. Their method effectively captures complex relationships & dependencies among IoT devices and their communications, thereby enhancing the robustness and accuracy of cyber attack recognition in IoT environments.

2.3 Lightweight Cyber Attack Detection Model

Much effort has been expended over the past ten years to address security problems are connected to detecting cyber attacks in IoT systems. Many of these anomaly-based ADS systems are developed employing machine learning (ML) & deep learning (DL) techniques to enable intelligent cyber security decisions. DL techniques work on different types of dataset utilized for testing and training predictive ADS models, so all features/records in the dataset are important when training/testing classification/detection models (Alsulami et al., 2022).

Thus, feature preprocessing and data engineering formed the central part of the DL-based ADS model. This has been crucial in enabling the raw data gathered within the ecosystem of IoT to be analyzed and predicted further (Alsulami et al., 2022). They found highly encouraging results by applying model of deep neural network on two chosen IoT gateways, with an average increase in network bandwidth of less than 30 kb/s and a CPU consumption increase of only 2%. On the two datasets that were used, the ML models were able to show almost 93% detection accuracy and 92% f1-score (Rajendran et al., 2019).

(Wang et al., 2023) employed dynamic quantization to trim specified model cell structure, thereby reduce the computational complication on devices while maintaining accurate detection ability.

IoT attack and malware detection techniques are widely suggested in the literature currently in publication for specific use cases of cyber attack detection (Bhandari et al., 2023; Boppana and Bagade, 2023), malicious traffic detection (Lin, Xu, and Xiao, 2022), anomaly detection (Ullah and Mahmoud, 2022), and botnet detection (Awajan, 2023) using various AI methods: deep neural network (DNN) (Boppana and Bagade, 2023; Abdalgawad et al., 2022), and cyber threat intelligence (Bhandari et al., 2023).

Additionally, the study by (Alsulami et al., 2022) investigated the IoTID20 dataset with five different subsets with typical traffic: Normal, Mirai, DoS attack, MITM, and Scan attack. Nevertheless, The writers didn't look at the malware attacks that we addressed in this work. A different study (Bhandari et al., 2023) used a multi-stage classification method by applying clustering with oversampling to analyze the same dataset and predict the intrusion's existence. It merely recognizes the presence of attacks; it cannot, however, forecast the kind of the attacks. When feature/data engineering techniques are not used, it can often be more challenging to extract pertinent features using DL algorithms alone. Industry businesses and researchers have created techniques to classify the data and identify assaults by identifying important characteristics.

IoT traffic usually requires a huge amount of memory space and data capacity. However, putting cyber attack-detection and classification systems by high-performance deep learning based on IoT devices can be difficult in some situations due to resource limitations. Taking into account the computational and power limitations of edge devices (IoT), (Sudqi Khater et al., 2019) provide a vector space representation of a lightweight IDS based on Multilayer Perceptron (MLP) paradigm. The Australian Defense Force Academy Windows Dataset and the Australian Defense Force Academy Linux Dataset, which are modern system call datasets containing various application exploits and attacks, were utilized to evaluate the performance of the given IDS. For complex high-dimensional hidden layer data, (Sanju, 2023) developed a lengthy short-term memory autoencoder technique to produce low-dimensional intrinsic spatial representations of features.

Employing a hybrid BiLSTM technique, (Ibibo, 2023) achieved the highest possible data reduction rate is 91.89% for classifying cyber attacks, both in binary and multi-classification studies, using the BoT-IoT classical botnet dataset. Additionally, (Ibibo, 2023) proposed a unique lightweight cyber attack identification algorithm based on ABC (Artificial Bee Colony) for analyzing Sybil assaults in cellular low-power, distorted networks, leveraging bioinspired principles. Through their research, (Ibibo, 2023) showcased the accuracy, sensitivity, and efficacy of this lightweight cyber attack-detection method against the three distinct kinds of Sybil assaults.

(Pham, Seo, and Chung, 2020) introduce a novel lightweight cyber attack identification system that employs a unique approach of transforming raw data of network traffic into representation images. Initially, they analyze the characteristics of abnormal network traffic using the CSE-CIC-IDS2018 dataset and subsequently devise methods to effectively represent these traits as image data. Utilizing a CNN-based detection model, the system identifies malicious traffic patterns embedded inside the picture data. To validate the practicality of their lightweight cyber attack-detection system, (Pham, Seo, and Chung, 2020) conduct three simulations using two datasets containing encrypted traffic with current network attack scenarios. The experimental findings highlight the capability of their proposed cyber attack detection system to achieve 95% accuracy with a reasonable detection time, utilizing relatively small training data sizes.

(Khater, 2021) delineate lightweight research into three distinct processes: data preprocessing, then model selection, design, and analysis of performance. They employed an enhanced method of representing vector spaces known as the n-gram method for feature extraction, while employing selection of feature based on communal information

to minimize feature dimensionality. In contrast, in (Sanju, 2023) used advanced meta-heuristics technique with an ensemble of RNN to improve the attack identification in IoT.

In their research, (Alzubi et al., 2022) introduced a unique hybrid dimensionality reduction strategy by integrating the GWO with genetic algorithms (GA). Tailored for real wireless datasets, this hybrid method aims to pinpoint the most pertinent features within wireless traffic data. Evaluating its efficacy, they employed support vector machines (SVM) as classifiers in experimental settings, with results showcasing the algorithm's ability to reduction by 68 dataset dimensionality while also decreasing computation time.

Similarly, (Lahasan and Samma, 2022) proposed the self-encoder approach. Through experimental analysis on an IoT cyber attack-detection dataset, they demonstrated that the model with lightweight capability exhibits high cyber attack-detection accuracy, surpassing most classical optimizers.

From the various lightweight IoT cyber attack detection scheme approaches discussed above, it can be deduced that the majority of studies focus on two main aspects: dimension reduction of features for dataset preprocessing and model lightweighting to conserve memory capacity & Internet of Things device processing power. Furthermore, researchers have begun integrating optimization algorithms help overcome these obstacles and improve the efficacy of IoT settings' cyber attack identification systems.

TABLE 2.2: Summary of research publication related to Lightweight Intrusion detection systems

Ref.	Datasets	Algorithms	Results	Cyber-attack	Limitations
(Zhao et al., 2021b)	USW-NB15	PCA-LNN	86.11%	Norm, Dos	No hyper parameter selection
	BoT-IoT		96.14%	DDos, Dos Scanning	Low detection and classification rate
(Wang et al., 2024)	CIC-IDS2017	Knowledge distillation	99%	Norm, Dos DDos	Model generalizability of detection system is under question
	TON-IoT		99%	Port scan XSS, MiTM Dos, DDos	Low detection rate classification rate
(He et al., 2023)	TON-IoT	Session merging & feature engineering	99.4%	Dos, DDos XSS, MiTM Backdoor Ransomware	Small no of attacks Does not use large and sophisticated dataset Model generalizability of detection system is under question
(Murugesan and Maheswaravenkatesh, 2024)	CICIDS2018	Convolutional & neural network DNN	98.7%	Benign, DDos Bot Infiltration SQL-injection Brute force	Algorithm optimization Does not use large and sophisticated dataset
(Wang et al., 2023)	CIC-IDS2017	Post training Quantization	99.5%	Norm, Dos Port scan, Bot	Model generalizability is under question
	CIC-IoT2023		93.13%	Recon, Mirai, DoS, web-based brute force, DDoS	Higher model size, high number of parameters and higher FLOPs

2.4 Summary and Research Gap

Through the review of various papers and research works, several significant gaps have come to light.

The First critical gap lies in dataset preprocessing part. Effective feature engineering mechanisms are essential to reduce dataset dimensions and optimize it for IoT models. The accuracy and performance of these models are heavily contingent upon the dataset's quality. Feature engineering involves the removal of unnecessary features and the extraction of new ones to bolster accuracy. Eliminating irrelevant features is crucial to mitigate overfitting and under-learning, which can significantly impact classifier performance and execution time. An exhaustive data preprocessing with improved feature engineering architecture is necessary to enhance system performance. Thus, our approach entails providing such a framework, complete with meticulous development, validation environments, and configurations.

Secondly, algorithm complexity stands out as the primary gap. Many existing deep learning solutions are excessively intricate and prone to overfitting. Given the resource-constraints of IoT devices, algorithms must be lightweight for real-time use. Therefore, we aim to develop lightweight algorithms tailored for IoT deployment. These algorithms was optimized to operate within the limited memory and computational power of IoT devices.

Thirdly, while much research emphasizes identification and categorization accuracy, the efficiency of IoT systems in detecting cyber attacks is often overlooked.

In summary, addressing these gaps necessitates a comprehensive approach that encompasses model complexity, efficiency, dataset management, and algorithm optimization. By focusing on these aspects, we aim to develop more effective and efficient IoT-based detection systems.

Chapter 3

Methodology

3.1 Research Methodology

With this study, we employed experimental and machine learning research methodologies to advance academic research in cyber attack detection. By integrating these approaches, we aimed to enhance the accuracy and efficacy of identifying potential threats.

3.2 Dataset and Preprocessing

With this experiment, we utilized two widely acknowledged public datasets for cyber attack identification, specifically selected to function as benchmark standard datasets for IoT applications. Our focus was on conducting cyber attack identification experiments to verify the efficiency of our proposed cyber attack identification model prepared for IoT environments. Below, we present a brief illustration of the datasets employed, along with comprehensive details regarding the preprocessing stages undertaken.

3.2.1 Dataset Overview

The process of data collection involves systematically gathering information or data concerning specific variables within a dataset in an organized manner. This enables researchers to effectively address research questions, test hypotheses, and evaluate outcomes. In the context of this research, the variables of interest pertain to cyber attacks targeting data records within IoT computing environments. The chosen dataset was utilized to assess the Efficacy of an Attack Detection System (ADS) for IoT systems, as well as to apply various data engineering techniques aimed at enhancing the learning

and validation processes. The study will encompass the examination of two prominent global datasets relevant to IoT attacks, including CIC IDS2017 and the CICIoT2023 dataset.

To enhance the efficacy of cyber attack recognition for IoT, in this research improved feature engineering techniques was applied to enhance the learning and validation processes. Preprocessing steps was undertaken to refine the dataset, particularly focusing on enhancing the multi-classification accuracy of label, category, and features.

CIC IDS2017 dataset is facilitated by the Canadian Institute of Cyber Security and captured through CICFlowMeter software, offers a comprehensive view of network traffic characteristics spanning five days, from Monday until Friday evening (Sharafaldin, Lashkari, Ghorbani, et al., 2018). This dataset used as a crucial resource for research in system security and cyber attack identification, encompassing a wide array of attack scenarios such as Botnet, Brute Force, DoS, DDoS, Infiltration, SQL Injection, Port scan, and XSS. It fulfills the criteria outlined in the dataset assessment method. In our study, we specifically focus on the subset of data collected and generated on a Friday, tailored for the purpose of IoT attack detection. For a comprehensive understanding, detailed explanation of CIC IDS2017 datasets utilized are provided in Table 3.1. Also for comparsion purpose the same dataset used as used in (Wang et al., 2023).

TABLE 3.1: CIC IDS2017 dataset attack distribution.

Category	Training Data	Testing Data	Validation Data
BENIGN	265326	82785	66211
Bot	1263	405	298
DDoS	81778	31867	20598
PortScan	101709	25381	25413
Total	450076	140649	112520

CICIoT2023 dataset is developed by Neto and colleagues in 2023, presents an extensive compilation of IoT attacks. It encompasses 105 genuine IoT devices organized into diverse topologies, simulating a smart home setting. Within this dataset, thirty-three unique attacks are labeled into nine groups: Brute force, DNS spoofing, DoS, DDoS, Recon, and Mirai. These attacks were performed, recorded, and collated within the IoT environment. Notably, the dataset includes Recon attacks, which involve gathering comprehensive target data from the IoT architecture, and Mirai attacks, characterized by large-scale DDoS assaults on IoT devices. These attack categories represent both typical and innovative kinds of IoT traffic attack. This research concentrate on these

cyber attack categories for experimental purposes. Further details regarding dataset can be observed in Table 3.2.

TABLE 3.2: CICIoT2023 dataset attack distribution.

Category	Training Data	Testing Data	Validation Data
BENIGN	120779	37758	30418
Dos-TCP_flood	294903	92116	73383
DDos-TCP_flood	98493	30939	24563
DNS-Spoofing	344	95	83
Recon-PortScan	19892	6104	4927
Recon-OSScan	33871	10669	8489
Mirai-greeth_flood	82792	25766	20907
Mirai-udpplain	8952	2896	2243
Mirai-greip_flood	11010	3356	2746
Total	671036	209699	177679

3.2.2 Data Preprocessing

It is vital to preprocess datasets before utilizing it in deep learning architecture. Data sourced from real-world environments typically harbor numerous errors and irregularities, necessitating meticulous cleaning procedures. For instance, when encountering string values within the dataset, it becomes essential to convert them into numerical format prior to deep learning training. Through preprocessing, the objective is to furnish the algorithm with refined and structured data, ultimately optimizing the model's efficiency and effectiveness in producing accurate outcomes.

Data preprocessing phase plays a significant role in preparing data for deep learning techniques. It involves several steps to change real world sensor data into a more suitable format for neural networks. These steps typically include data cleaning for handling missing values and outliers, label encoding to convert categorical variables into numerical format, feature engineering to create new features or transform existing ones, normalization to scale features to a similar range, and data splitting into 80 % training, 20% testing , and then further divide the training set to training and validation 80% and 20% respectively. Each of these steps contributes to enhancing the quality and usability of the data for subsequent deep learning tasks.

To align with the format of input required by the neural network for the cyber attack identification architecture proposed in this research, the experimental dataset necessitates preprocessing. This preprocessing procedure for the attack identification system comprises five distinct stages, outlined as follows:

3.2.2.1 Dataset Cleaning

Prior to model training, it is essential to conduct a rigorous examination of datasets to identify any instances containing missing or undefined values, ensuring both the accuracy of the input data and the reliability of the resulting model. In this study, pre-processing was conducted using the PyTorch framework along with Python's built-in libraries. We analyzed the CIC IDS2017 and CICIOT2023 datasets and identified missing and infinite values, prompting a cleaning process. Two approaches were evaluated: (1) a three-step method involving converting infinite values to NaN, imputing missing values with column means, and removing any remaining NaN or infinite values, and (2) direct removal of rows with missing values. As missing data represented only 0.000284% of the total and had no impact on model accuracy according to our experiment, we opted for direct removal to streamline preprocessing without affecting performance.

3.2.2.2 Label Encoding

Utilizing numerization is a recognized technique for handling categorical data, particularly in scenarios where memory efficiency and computational speed are crucial. Unlike one-hot encoding, which can be memory-intensive and time-consuming, numerization assigns a distinctive numeric value to each category within a categorical feature.

In context of deep learning method, it is imperative that both input & output values are represented as integers. This necessitates encoding categorical features or it is numerization of the feature. Within the dataset being employed for this research, there exists a column with symbolic class feature, which is essentially a categorical variable denoting different attack types.

For binary experiments, the categorical data is encoded into zeros and ones, where zero signifies the normal class and one indicates the attack class. Meanwhile, in multiclassification experiments, the numeric values for the categorical labels are incremented accordingly. In CICIOT2023 dataset, Benign class is represented by 0, Dos-TCP_flood by 1, DDos-TCP_flood by 2, Mirai-greeth_flood by 3, Mirai-udpplain by 4, Mirai-greip_flood by 5, Recon-PortScan by 6, Recon-OSScan by 7, and DNS-Spoofing by 8.

By employing numerization, we ensure that the categorical features are appropriately converted into numerical values, facilitating seamless integration into deep learning models. This approach not only optimizes memory usage but also streamlines computational efficiency, thereby enhancing the overall effectiveness of the training of the model.

3.2.2.3 Feature Engineering

In addressing the issue arising from the high-dimensional nature of the original dataset's and redundancy of features, feature engineering becomes a pivotal aspect of this research endeavor. Effective feature engineering involves identifying and transforming real world sensor data into model understandable features that enhance model performance. Techniques such as feature scaling, normalization, and extraction of relevant features play a pivotal role in this process. Additionally, the selection of appropriate feature dimensionality reduction methods is paramount to mitigate computational complexities and improve model efficiency.

Feature engineering involves the process of either eliminating redundant features or extracting new ones from existing features, aimed at enhancing the performance using machine learning (Derhab et al., 2020)). Redundant records were identified and eliminated from the dataset. Since each dataset possesses its unique feature set, it's crucial to eliminate irrelevant features dimensions that are not critical to the output label. Retaining such insignificant features can lead to overfitting or underfitting, thereby adversely affecting both the execution time and classifier performance.

Employing Redundancy-Adjusted Logistic MIFS (RAL-MIFS) enhances feature engineering for Cyber Attack Detection Systems (ADS) by refining Mutual Information Feature Selection (MIFS). By adjusting for feature redundancy and leveraging logistic weighting, RAL-MIFS prioritizes the most informative features. This approach improves the efficacy and accuracy of ADS, ensuring better detection of attacks in complex datasets.

The original dataset's high-dimensional features pose challenges for the proposed light-weight attack detection model. To address this, selecting an appropriate feature dimension reduction method is crucial. In this investigation, we propose RAL-MIFS combined with two-stage IPCA approach to solve the issue of high-dimensional features. The two-stage IPCA method involves performing IPCA in sequential stages, each stage refining the dimensionality reduction process further. This approach efficiently transforms high-dimensional features into lower-dimensional representations through mapping functions, thereby facilitating dimensionality reduction. By employing a two-stage IPCA approach, we target to optimize the reduction of feature dimensions while preserving the relevant information necessary for accurate attack detection.

Through the usage of feature engineering techniques and the innovative RAL-MIFS + two-stage IPCA method, the aim is to enhance the features quality and relevance used in our light-weight attack detection model. By transforming the data from sensors

which is raw into a more meaningful representation and refining the dimensionality reduction process iteratively, we can improve the model capacity to discern patterns and anomalies effectively. This ensures that our model is not only efficient but also capable of accurately identifying and classifying various attack types within the dataset.

3.2.2.4 Normalization

It is a technique mainly used in preprocessing of dataset for DL algorithms. The aim of normalization is to change the numeric column values in a data to a uniform scale while keeping changes in value bounds. Each feature of CIC IDS2017 dataset, and CI-CIoT2023 dataset has unique values. A certain amount of feature values reach into the thousands, and some may contain negative figures that degrade the model performance. To solve this challenge, the dataset was normalized between zero and one via min max technique, as shown in equation 3.1. The data is transformed into a NumPy array and resized to match the total number of records and input features utilizing Python's NumPy library.

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (3.1)$$

3.2.2.5 Data Splitting

Break apart the data to train, test, & validation sets is the widely used preprocessing steps used to investigate the DL models' performance. When dealing with balanced data, it is advantageous to utilize random sampling due to its simplicity and ability to represent the entire dataset. Random sampling is effective in creating subsets that are representative of the whole dataset without introducing systematic bias. Random sampling was employed to partition the dataset into training, validation, and test sets, thereby maintaining a representative distribution across all subsets.

After we normalize the dataset, each matrix value will have a value between zero and one. For the data partitioning procedure, data is randomly divided into 80 % training, 20% testing, and then further divide the training set to training and validation 80% and 20% respectively. We will use across-validation technique as a validation scheme for our research. Finally, data was fed to deep learning model.

3.3 Proposed Lightweighting Architecture

In recent years, the demand for lightweight and efficient cyber attack detection systems has surged due to the proliferation of IoT devices of constrained processing capabilities. However, existing models often fall short in meeting these requirements, as they entail high computational costs and are unsuitable for widespread deployment. The emergence of CNN, BiLSTM and DNN has shown promise in addressing these challenges, yet research in this area remains limited. While CNN-based classifiers excel in managing diverse datasets, their application in attack detection has only recently gained traction, outperforming traditional approaches (Papaioannou et al., 2022). To capitalize on this advancement, our study proposes an innovative model that integrates a DNN with BiLSTM, showcasing exceptional performance in identifying various network anomalies and malicious assaults. Despite the outstanding performance of BiLSTM & DNN, our hybrid approach offers a robust solution for lightweight cyber attack recognition in IoT environments (Rani et al., 2022).

Plenty of current techniques for cyber attack recognition and categorization often rely on simplified machine learning techniques due to resource-constraintness associated with deploying cyber attack identification systems for IoT devices. However, there exists an upper limit on detection accuracy, and overlooking attacks could lead to significant property damage. In this investigation, we propose a lightweight deep learning-based neural network cyber attack identification and classification scheme that incorporates two stage feature dimension reduction and advanced dynamic model quantization to streamline the detection model complication. The outcome is a model that maintains low computational complication while achieving best detection accuracy.

The following are the phases of the suggested method for detecting IoT attacks: First, as explained in Subsection 3.2.2, the IoT cyber attack detection and classification dataset is preprocessed through data encoding or numerization, data cleaning, feature engineering (RAL-MIFS), and two stage IPCA algorithm. The preprocessed data was divided to test, training, and validation category. The test set confirms the final model's classification performance, while the training & validation data are utilized for training the model and weight fine tuning. After that, training & validation data are fed into DNN-BiLSTM model and quantization aware training was performed.

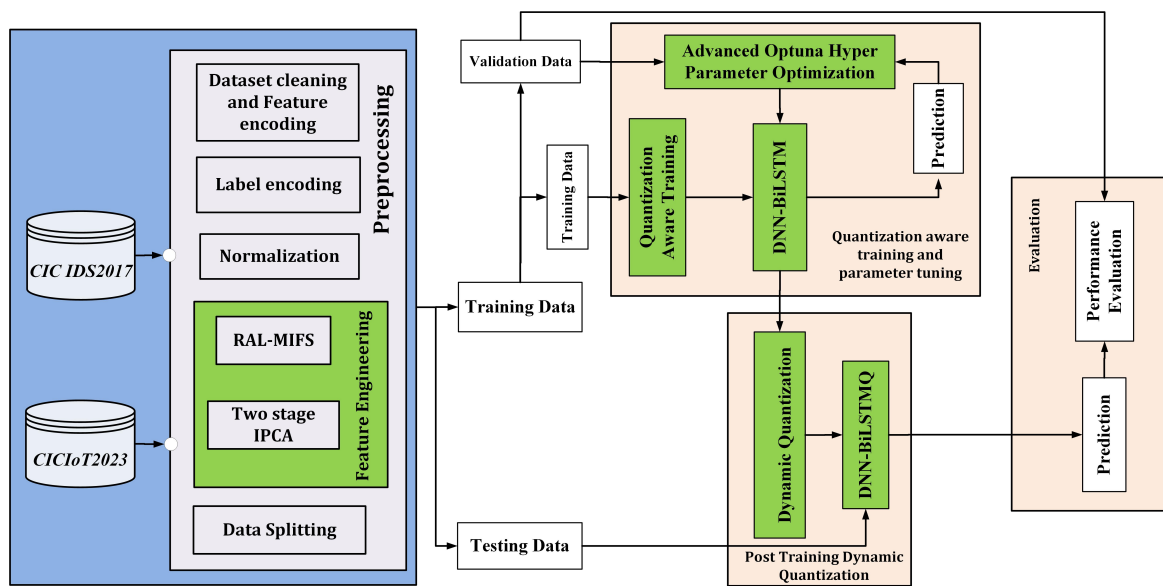
To determine the proper parameters of the proposed system, the hyperparameters of the model are then optimized using the optuna optimization techniques. The trained optimal model is then subjected to advanced dynamic quantization (Quantization Aware

Training and Post training dynamic quantization) by means of structural unit specifications. The quantized DNN-BiLSTMQ model is generated, and the ultimate identification performance of the final model is assessed through rigorous testing of the reduced size quantized model with the designated test set. The proposed general architecture of lightweight deep learning based IoT cyber attack identification and categorization method is illustrated in Figure 3.1, comprising four main modules: data preprocessing, quantization aware model training & weight fine tuning, post training model dynamic quantization, and model categorization testing. This comprehensive architecture aims to enhance intrusion detection efficiency while minimizing computational resources, ensuring effective protection against cyber attack in IoT environments.

Ensuring accurate evaluation of the deep learning model's performance necessitates proper data splitting into train, test, and validation sets. To address potential issues arising from unbalanced datasets, we employ a stratified splitting method. Following normalization, data is divided into 80 % training, 20% testing , and then further divide the training set to training and validation 80% and 20% respectively. Cross-validation serves as our validation scheme before feeding data into the deep learning model.

The proposed architecture for IoT cyber attack identification and categorization aims to address the limitations of current methods by leveraging deep learning techniques. Through preprocessing, dataset division, model training, quantization, and optimization, our approach maintains high detection accuracy while reducing computational complication. This ensures effective detection of attack threats while optimizing resource utilization.

FIGURE 3.1: Proposed lightweight hybrid deep learning architecture



The research focuses on model lightweighting for IoT cyber-attack detection. In the subsequent sections, the proposed architecture will be described in detail. Beginning with data preprocessing, improved feature engineering was applied, which includes RAL-MIFS and a two-stage IPCA process, as detailed in Section 3.4.

Following this, we will develop a hybrid DNN-BiLSTM model, which comprises a DNN and BiLSTM architecture for intricate feature extraction as described in Section 3.5. The training of this model will incorporate quantization-aware training.

Then the developed hybrid model parameters will be optimized using advanced optuna parameter optimization, as detailed in Section 3.7, to achieve the best accuracy and while reducing model size.

Then to create lightweight model, we will apply advanced dynamic quantization, which integrates quantization-aware training with dynamic quantization after training, as described in Section 3.6.

This comprehensive approach aims to achieve a lightweight yet highly accurate model by systematically optimizing each stage of the development process.

3.4 Improved Feature Engineering

When real time sensor and system data directly inputted to neural network, unimportant redundant attributes in the real time sensor and system data typically make learning process harder, which definitely has a adverse impact on the efficacy of the cyber attack identification of entire system. The determination of relationships between input and target variables may necessitate a considerable number of instances, particularly in scenarios where the input data contains ample information but suffers from high dimensionality. Feature reduction presents a viable solution to these challenges, comprising two primary techniques: feature selection and extraction. Feature selection involves the identification and retention of pertinent inputs from the raw dataset, whereas extraction of feature entails the derivation of new parameters from the original dataset, encapsulating the most pertinent information relevant to the output variable.

3.4.1 Redundancy-Adjusted Logistic MIFS

Employing Redundancy-Adjusted Logistic MIFS (RAL-MIFS) enhances feature engineering for cyber attack Identification Systems by refining MIFS. By adjusting for feature redundancy and leveraging logistic weighting, RAL-MIFS prioritizes the most informative features. This approach improves the efficacy & accuracy of ADS, ensuring better detection of attacks in complex datasets.

As stated in(Hoque, Bhattacharyya, and Kalita, 2014) , MIFS is a popular system for feature identification that can effectively choose important features aside from data distribution, which makes best for early attack detection architecture where data have no sufficient attack patterns. Unlike existing MIFS that calculate k using Equation (3.2), we proposed RAL-MIFS that calculate the redundancy using Equation (3.3). Rather than updating k value linearly, the RAL-MIFS gradually rise weights whenever a new attributes is appended to the dataset.

$$k = \frac{1}{|S|} \quad (3.2)$$

$$k = \frac{1}{1 + e^{-\left(\frac{|S|}{|F|}\right)}} \quad (3.3)$$

Where: k is the redundancy coefficient, $|S|$ is count of features in the choosen dataset and $|F|$ is the number of features in the original dataset.

Principal Component Analysis has emerged as the prevailing method for feature reduction in prior research on attack detection systems, offering enhanced lightweight

performance via its low-complexity linear transformation approach. Variants of PCA have been extensively applied for reduction of feature dimensionality in attack detection datasets, reflecting its ongoing relevance. However, a common challenge lies in the batch processing nature of conventional PCA algorithms, which necessitates concurrent collection of all data observed for computation and often suffers from poor computational efficiency. This issue is particularly pronounced in IoT environments in which real-time monitoring is imperative. In such contexts, full-fledged training data may not be acquired at once, and gradual increase in data traffic may lead to subtle alterations in cyber attack behavior from the dataset. Consequently, features extracted from initially provided training samples may lack persuasiveness, underscoring the desire for more adaptable and efficient feature reduction techniques in dynamic IoT settings.

Selected Features from the CIC-IDS2017 Dataset: The RAL-MIFS algorithm was employed to filter and prioritize the most informative features from the CIC-IDS2017 dataset, ensuring that they significantly contribute to the analysis and classification tasks. The selected features are encoded as numerical values.

TABLE 3.3: Selected features CIC-IDS2017

CIC-IDS2017 Selected Features
52, 4, 63, 40, 0, 42, 41, 8, 53, 6, 65, 5, 66, 12, 54, 34, 55, 10, 67, 14, 35, 1, 69, 18, 37, 36, 20, 23, 15, 21, 16, 39, 62, 2, 11, 64, 3, 68, 17, 7

Selected Features from the CICIoT2023 Dataset: The RAL-MIFS algorithm was employed to filter and prioritize the most informative features from the CICIoT2023 dataset, ensuring that the selected features significantly contribute to the analysis and classification tasks. TThe selected features are encoded as numerical values.

TABLE 3.4: Selected features CICIoT2023

CICIoT2023 Selected Features
39, 35, 38, 33, 41, 36, 34, 1, 2, 4, 5, 0, 45, 40, 26, 3, 37, 42, 43, 18, 44, 17, 11, 27, 15, 20, 31, 32, 14, 16, 19, 10, 8, 9, 24, 7, 25, 28, 21, 30

3.4.2 Two stage IPCA

This study also introduces the two-stage IPCA for Detecting and classifying cyber attacks, as a improvement for the PCA algorithm. IPCA, an incremental learning-based reduction of feature dimensionality method suggested by (Ozawa, Pang, and Kasabov, 2010), facilitates the incremental updating of feature vectors and values. Below is a concise outline of the reduction of dimensionality principle employed by the IPCA algorithm (Ozawa, Pang, and Kasabov, 2010). The high dimensional data samples will be denoted as $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m] \in \mathbb{R}^{n \times m}$ in an n -dimensional space, and mean of $\mathbf{X}$ will be computed from that. $\mathcal{M}_{\mathbf{X}}(\bar{\mathbf{X}}, \mathbf{U}_k, \mathbf{\Lambda}_k)$ is the eigenspace model of the raw dataset. Consider the new sample data $\mathbf{Z} \in \mathbb{R}^n$ will be upgraded to the the sample data of original; then, $\bar{\mathbf{X}}$ the initial model eigenspace will be upgraded to $\bar{\mathbf{X}}'$, where

$$\bar{\mathbf{X}}' = \frac{1}{N+1} \sum_{i=1}^N (\bar{\mathbf{X}}N + \mathbf{Z}) \in \mathbb{R}^n. \quad (3.4)$$

Where $\mathbf{U}_k$ is $n \times k$ matrix, and k represents the feature dimension space, with the matrix columns representing the data's feature vectors. $\mathbf{\Lambda}_k$ represents $k \times k$ dimension matrix with diagonal members that match to the data's eigenvalues. The decision on updating feature vectors and values in the vector feature space hinges on if the newly updated data samples introduce previously absent energy in the existing feature space, as determined by the following equation:

$$g = \mathbf{U}_k^T (\mathbf{Z} - \bar{\mathbf{x}}) \quad (3.5)$$

where $\bar{\mathbf{x}}$ is the mean vector of the new data.

$$h = (\mathbf{Z} - \bar{\mathbf{x}}) - \mathbf{U}_k \cdot g \quad (3.6)$$

$$\hat{h} = \begin{cases} \frac{h}{\|h\|}, & \text{if } \|h\| > \eta, \\ 0, & \text{otherwise.} \end{cases} \quad (3.7)$$

In the context of our investigation, the residual vector h belonging to the real space $\mathbb{R}^n$ denotes a crucial element. The determination to adjust the feature dimensional space k is contingent upon the calculation of norm $\|h\|$, which prompts an increment by one if it exceeds the predefined threshold η . Conversely, if the norm $\|h\|$ remains below η , the dimension of vector feature space retains its original value of k . In scenarios where an adjustment in the feature space dimensionality is warranted, the matrices $\mathbf{U}_k$ and $\mathbf{\Lambda}_k$

undergo transformation to $\mathbf{U}'_{k+1}$ and $\mathbf{\Lambda}'_{k+1}$ respectively, as governed by equations:

$$\mathbf{U}'_{k+1} = [\mathbf{U}_k, \hat{h}] \cdot R \quad (3.8)$$

$$R\mathbf{\Lambda}'_{k+1} = \left\{ \frac{n}{n+1} \begin{bmatrix} \mathbf{\Lambda}_k & 0 \\ 0^T & 0 \end{bmatrix} + \frac{n}{(n+1)^2} \begin{bmatrix} gg^T & \gamma g \\ \gamma g^T & \gamma^2 \end{bmatrix} \right\} R \quad (3.9)$$

However, if the vector feature space dimension will not expanded, then $\mathbf{U}_k$ and $\mathbf{\Lambda}_k$ will be added to $\mathbf{U}'_k$ and $\mathbf{\Lambda}'_k$:

$$\mathbf{U}'_k = \mathbf{U}_k R \quad (3.10)$$

$$R\mathbf{\Lambda}'_k = \left\{ \frac{n}{n+1} \mathbf{\Lambda}_k + \frac{n}{(n+1)^2} gg^T \right\} \quad (3.11)$$

For the second stage of IPCA, the updated $\mathbf{U}'_k$ and $\mathbf{\Lambda}'_k$ from the first stage are used. Let R' be the rotation matrix for the second stage. Then, the updated in the feature space vector, feature vectors and values are given by:

$$\mathbf{U}''_{k+1} = [\mathbf{U}'_k, \hat{h}'] \cdot R' \quad (3.12)$$

$$R'\mathbf{\Lambda}''_{k+1} = \left\{ \frac{n}{n+1} \begin{bmatrix} \mathbf{\Lambda}'_k & 0 \\ 0^T & 0 \end{bmatrix} + \frac{n}{(n+1)^2} \begin{bmatrix} g'g'^T & \gamma'g' \\ \gamma'g'^T & \gamma'^2 \end{bmatrix} \right\} R' \quad (3.13)$$

where $\hat{h}'$ is obtained similarly as $\hat{h}$, g' is computed as $g' = \mathbf{U}'_k{}^T(\mathbf{Z} - \bar{\mathbf{x}}')$, and γ' is a scalar calculated from g' .

Two-stage IPCA contribute to improving the robustness and reliability of data analysis and machine learning models in diverse real-world applications.

3.4.3 RAL-MISF + Two Stage IPCA Pseudocode

Algorithm 1 Two-stage IPCA + RAL-MIFS Feature Selection

Input: Dataset $X \in \mathbb{R}^{n \times m}$, New data $Z \in \mathbb{R}^n$, IPCA components $U_k \in \mathbb{R}^{n \times k}$, Eigenvalues $\Lambda_k \in \mathbb{R}^{k \times k}$, Threshold η , Number of samples N , Number of components k

Output: Updated mean $\bar{X}'$, Updated IPCA components $(U''_{k+1}, \Lambda''_{k+1})$, Selected features X_{selected}

```

1: procedure TWO STAGE IPCA + RAL-MIFS
2:   Step 1: Feature Selection using RAL-MIFS
3:    $G = [], S = [] \rightarrow$  Initialize empty lists
4:   for each feature  $f_i \in F$  do
5:      $g_i = \text{MI}(f_i, C) \rightarrow$  Calculate MI and  $G = G \cup \{g_i\} \rightarrow$  Store MI in  $G$ 
6:   end for
7:    $g_k = \max(G) \rightarrow$  Select feature with highest MI and  $S = \{g_k\} \rightarrow$  Add feature with highest
   MI score to selected list  $S$ 
8:   while  $|S| < n$ :
9:     for each  $g_j \in G, s_m \in S$  do  $\rightarrow$  Calculate RAL-MIFS
10:       $L = \text{RAL-MIFS}(C; g_j, s_m) \rightarrow$  Compute RAL-MIFS score
11:    end for
12:     $s_d = \arg \max_{s_m \in S} \sum_{s_m \in S} L \rightarrow$  Select feature maximizing RAL-MIFS and  $G = G \setminus \{s_d\} \rightarrow$ 
    Remove selected feature from  $G$ 
13:     $S = S \cup \{s_d\} \rightarrow$  Add selected feature to  $S$  and  $X_{\text{selected}} = \{f_i \mid f_i \in S\} \rightarrow$  Selected features
14:   Step 2: First Stage IPCA
15:    $\bar{X} \leftarrow \text{mean}(X_{\text{selected}})$ 
16:    $\bar{X}' \leftarrow \frac{1}{N+1}(N \cdot \bar{X} + Z_{\text{selected}})$ 
17:    $g \leftarrow U_k^T (Z_{\text{selected}} - \bar{X}')$ 
18:    $h = (Z_{\text{selected}} - \bar{X}') - U_k \cdot g$ 
19:   if  $\|h\| > \eta$  then
20:      $\hat{h} \leftarrow \frac{h}{\|h\|}$ 
21:      $k \leftarrow k + 1$ 
22:      $U'_{k+1} \leftarrow [U_k, \hat{h}] \cdot R$ 
23:      $R\Lambda'_{k+1} \leftarrow \frac{n}{n+1} \begin{bmatrix} \Lambda_k & 0 \\ 0^T & 0 \end{bmatrix} + \frac{n}{(n+1)^2} \begin{bmatrix} gg^T & \gamma g \\ \gamma g^T & \gamma^2 \end{bmatrix} \cdot R$ 
24:   else
25:      $U'_k \leftarrow U_k \cdot R$ 
26:      $R\Lambda'_k \leftarrow \frac{n}{n+1} \Lambda_k + \frac{n}{(n+1)^2} gg^T$ 
27:   end if
28:   Step 3: Second Stage IPCA
29:    $h' \leftarrow (Z_{\text{selected}} - \bar{X}') - U'_k \cdot g'$ 
30:    $g' \leftarrow U'^T_k (Z_{\text{selected}} - \bar{X}')$ 
31:   if  $\|h'\| > \eta$  then
32:      $\hat{h}' \leftarrow \frac{h'}{\|h'\|}$ 
33:   else
34:      $\hat{h}' \leftarrow 0$ 
35:   end if
36:    $U''_{k+1} \leftarrow [U'_k, \hat{h}'] \cdot R'$ 
37:    $R'\Lambda''_{k+1} \leftarrow \frac{n}{n+1} \begin{bmatrix} \Lambda'_k & 0 \\ 0^T & 0 \end{bmatrix} + \frac{n}{(n+1)^2} \begin{bmatrix} g'g'^T & \gamma' g' \\ \gamma' g'^T & \gamma'^2 \end{bmatrix} \cdot R'$ 
38:   return  $\bar{X}', U''_{k+1}, \Lambda''_{k+1}, X_{\text{selected}}$ 
39: end procedure

```

As shown in the Algorithm 1 the original dataset undergoes feature selection using RAL-MIFS to identify the optimal features. RAL-MIFS leverages mutual information and redundancy adjustment through logistic functions to prioritize features that contribute most significantly to the variable of interest while minimizing redundancy. This step ensures that the most vital features are retained, enhancing the subsequent analysis.

The enhanced dataset, containing the selected features, is then input into the first stage of IPCA, resulting in an updated dataset. This updated dataset is subsequently fed into the second stage of IPCA, yielding the final updated dataset. By performing dimensionality reduction in two stages, this method effectively reduces dimensionality of dataset with minimal compromise on accuracy, making it convenient for handling large-scale datasets incrementally.

3.5 Hybrid DNN-BiLSTM Model

In the preprocessing phase, there is a crucial step of refining data to organize incoming packet data efficiently. Many dataset utilized for cyber attack detection contains numerous attributes. However, utilizing all these attributes for cyber attack identification without refinement can result to a decline in performance due to overfitting concerns.

To get control of the issue, this study proposes an innovative approach which is hybrid deep learning that integrates DNN with BiLSTM, as shown in Figure 3.2. This combination is designed to upgrade the model's capacity for the extraction of nonlinear feature while leveraging BiLSTM's inherent capacity to capture bidirectional long-distance dependencies.

The hybrid DNN BiLSTM model begins by quantizing the input data (reducing its precision) for potential efficiency gains and also to extract best features during training. . It then processes the data through two fully connected layers with ReLU activations. Next, the model reshapes the data and feeds it into a Bidirectional LSTM layer that captures information from the sequence in both directions. Finally, the model selects the output from the last time step of the LSTM and processes it through another fully connected layer to generate the final output. The model then dequantizes the final output back to its original precision. These quantization steps help minimize the model size & potentially improve inference speed on devices with limited resources.

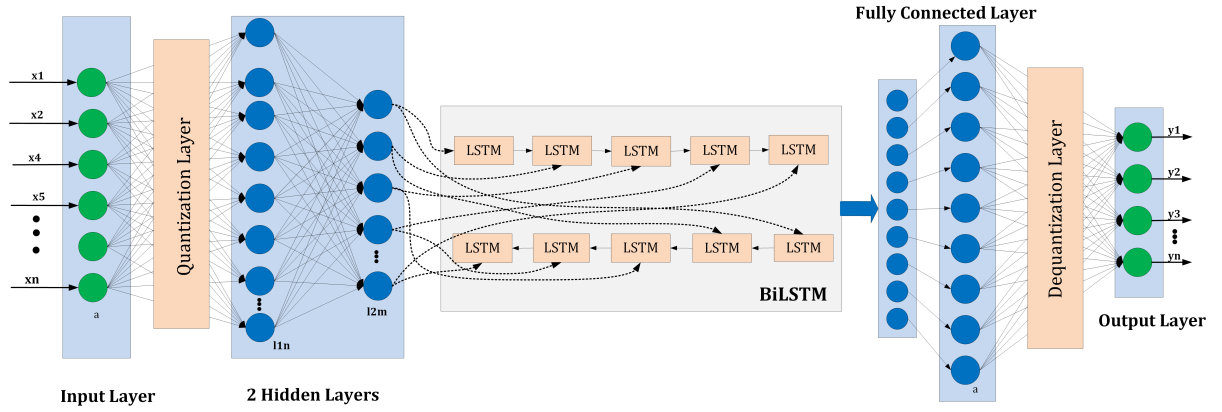
The model architecture seamlessly integrates DNN and BiLSTM layers, leveraging their unique strengths to improve detection accuracy. By integrating DNN layers, the model gains the capability to identify intricate patterns and anomalies within attack behaviors, thus enhancing its overall performance.

The incorporation of DNN and BiLSTM layers in the proposed model offers a synergistic advantage in detecting complex attack patterns. DNNs excel at capturing intricate patterns and high-level features within the data, enabling the model to discern subtle variations indicative of malicious behavior. Conversely, the BiLSTM layer enhances the model's temporal understanding by considering both past and future contexts, enabling it to capture long-range dependencies and temporal dynamics inherent in attack sequences. This amalgamation not only enhances the capacity for the extraction of nonlinear feature but also equips it with a comprehensive grasp of temporal relationships, thereby improving its efficacy in detecting sophisticated cyber threats.

Furthermore, the inclusion of BiLSTM layers ensures the preservation of bidirectional dependencies in sequential data, facilitating the detection of sophisticated attack patterns across varying time scales. Through adaptive training mechanisms and a robust

evaluation framework, the proposed model demonstrates enhanced learning dynamics, scalability, and efficiency, offering a promising solution for cyber attack-detection in dynamic and evolving cyber threat landscapes.

FIGURE 3.2: DNN_BiLSTM Model Structure



The model architecture differs for the two datasets as follows:

- **CIC IDS2017 Dataset**

$$x_n : n = 26$$

$$\text{Hidden Layer 1, } l_{n1} : n = 24$$

$$\text{Hidden Layer 2, } l_{m2} : m = 12$$

$$y_n : n = 4$$

- **CICIoT2023 Dataset**

$$x_n : n = 28$$

$$\text{Hidden Layer 1, } l_{n1} : n = 44$$

$$\text{Hidden Layer 2, } l_{m2} : m = 22$$

$$y_n : n = 9$$

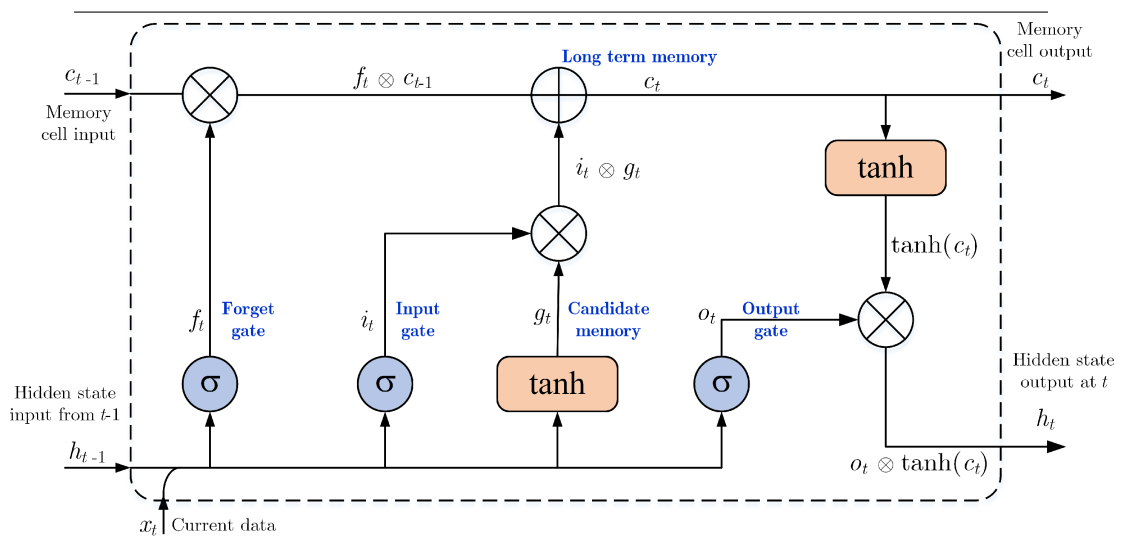
DNNs have significantly transformed the landscape of machine learning due to their capacity to extract intricate nonlinear features from data (Han, Kim, and Kim, 2022). A fundamental element contributing to their effectiveness is the usage of activation functions, which introduce nonlinear transformations to output of neurons of hidden layer. The sigmoid, Rectified Linear Unit (ReLU), and tanh, are among the most commonly employed activation functions.

The function called sigmoid is esteemed for its capability of mapping number to a range in the range zero and one, especially convenient for binary classification work. Tanh, another prevalent option, maps real numbers to a range spans from -1 and 1, providing flexibility advantageous for multiclassification pursuits. Meanwhile, ReLU is notable for its simplicity and efficiency; it outputs the input value when positive and zeros otherwise, enabling swift computations & finding broad application in DNN.

These activation functions has an important role in the processing of feature extraction, enabling DNNs to map inputs to nonlinear spaces effectively. By nesting multiple layers of these nonlinear transformations, DNNs can progressively learn and represent increasingly complex features functions within the data. This hierarchical representation permits DNNs to model intricate relationships and patterns, leading to enhanced performance across a wide array of tasks, ranging from image & speech recognition to natural language processing and beyond. Thus, the judicious selection and application of activation functions are fundamental to the efficacy of DL models.

While recurrent neural networks (RNNs) demonstrate adeptness in capturing temporal patterns, the persistent issue of addressing exploding gradient difficulties constrains their efficacy over prolonged periods, as evidenced in the research conducted by (Wang et al., 2023). To overcome this constraint, we opted for the specialized LSTM variant, Deep BiLSTM, which is an update of BiLSTM. This architecture demonstrates remarkable proficiency in analyzing sequential spatial features within the local environment, effectively capturing enduring patterns in spatial-temporal features. The detailed architecture of the Deep BiLSTM model is illustrated in Figure 3.3.

FIGURE 3.3: LSTM Structure (Mekruksavanich and Jitpattanakul, 2021)



The Deep BiLSTM model, introduced by (Mekruksavanich and Jitpattanakul, 2021), incorporates three essential gates: input gate, forget gate, and output gate. These gates play a vital role in regulating the flow of information within the LSTM unit. Specifically, LSTM can be accurately described as follows:

The LSTM cell consists of several components denoting different functionalities:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad (3.14)$$

Here, i_t represents the input gate, determining how many of current input x_t and the hidden state which is previous h_{t-1} present state of the cell should be stored.

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad (3.15)$$

f_t denotes the gate of forgetting, deciding how many of the prior cell state c_{t-1} should be retained.

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (3.16)$$

o_t is the output gate, determining how much information comes from the present cell state c_t should be output as the hidden state h_t .

$$\tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (3.17)$$

$\tilde{c}_t$ points the nominee memory cell, representing new nominee values for the cell state.

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \quad (3.18)$$

c_t is the present cell state, which is modified based on the gate which is input and the forgetting gate.

$$h_t = o_t \cdot \tanh(c_t) \quad (3.19)$$

Finally, h_t represents the present cell hidden state, which is determined by the output gate and the present cell state.

The bidirectional LSTM architecture tackles the problem of error accumulation by utilizing dual parallel LSTM layers. These layers process input sequences in both directions simultaneously. By doing so, this architecture significantly improves the model's

capability to capture sequential dependencies and enhance overall accuracy (Cai et al., 2022).

The feature extraction steps of BiLSTM will be expressed as equations.

$$h_t^{\rightarrow} = \tanh(W_{h_t}^{\rightarrow} x_t + W_h^{\rightarrow} h_{t-1}^{\rightarrow} + b_h^{\rightarrow}) \quad (3.20)$$

where $h^{\rightarrow}$ denotes the output result of the forward LSTM, h_t represents present cell hidden state, and h_{t-1} stands for output from prior cell hidden layer.

$$h_t^{\leftarrow} = \tanh(W_{h_t}^{\leftarrow} x_t + W_h^{\leftarrow} h_{t-1}^{\leftarrow} + b_h^{\leftarrow}) \quad (3.21)$$

where $h^{\leftarrow}$ denotes the output result of the reverse LSTM, h_t stands present cell hidden state, & h_{t-1} stands for prior cell hidden layer output.

$$h_t = h_t^{\rightarrow} + h_t^{\leftarrow} \quad (3.22)$$

This equation sums the forward and reverse LSTM outputs to acquire the final hidden state h_t of the present cell.

3.6 Model Lightweighting

According research by (Wang et al., 2022) there are several techniques possible to utilize to make deep learning algorithms more lightweight, and therefore more suitable for deployment on resource-constrained devices. Here are some well-known techniques: Quantization, pruning, knowledge distillation, and architecture optimization are among the techniques. Lightweighting techniques are detailed in Section 3.6.1.

3.6.1 Lightweighting Techniques

The first is Quantization which involves reducing the precision of the weights and activations in the network, typically from 32-bit floating point to 8-bit integers. This can significantly reduce the memory and computational requirements of the network without significantly affecting accuracy. The second is Pruning which involves removing unimportant connections from the network, typically those with small weights. This can significantly reduce the size of the network without significantly affecting accuracy. The third is Knowledge distillation which involves training a smaller "student" network to mimic the output of a larger "teacher" network. The student network can be much smaller compared to the teacher network, while still achieving comparable accuracy. The fourth is Architecture optimization which involves designing or selecting an architecture that is specifically optimized for the target device. This can involve techniques such as replacing convolutional layers with deep separable convolutions, or using smaller filter sizes. The fifth Quantized neural networks This involves training the network directly in quantized form, rather than quantizing it after training. This can lead to better accuracy than post-training quantization. Finally Model compression which is compressing the network by removing redundancies and decreasing parameters number. This can be done with techniques such as weight sharing, low-rank factorization, and tensor decomposition

For this thesis work based on the consideration to produce a model which is lightweight, short training, and time of inference, an advanced dynamic quantization (QAT + PTDQ) was used and as detailed in Section 3.6.2 .

3.6.2 Advanced Dynamic Quantization

Advanced quantization is a neural network optimization tactic that reduce model size & boosts inference speed by transforming weights and activations from floating-point to lower-bit integer formats during execution. This stratedy improves performance on edge devices and in environments with limited resources while preserving accuracy.

Advanced quantization is an optimization strategy utilized to decrease the bit width of model parameters from 32-bit floating-point to achieve better resolution and efficiency. It is significantly enhanced by employing advanced dynamic quatization a combined approach of quantization-aware training (QAT) and post-training dynamic quantization (PTDQ) techniques. QAT integrates the considerations of quantization effects into the process of training, guaranteeing that the model adapts effectively to the constraints imposed by lower bit-width representations. This proactive approach results in models that are inherently more robust to quantization, leading to better performance post-quantization compared to conventional methods.

By incorporating QAT to the training process, the model will becomes more resilient to variations in data distribution and noise, leading to enhanced generalizability across diverse datasets. QAT enables the earning capability to the model robust features that are less sensitive to quantization-induced distortions, thereby facilitating better adaptation to unseen data during inference.

Furthermore, the utilization of post-training dynamic quantization techniques ensures that the model's performance remains consistent across different deployment scenarios. These techniques optimize the quantization process while preserving crucial information, thus minimizing the risk of accuracy degradation on real-world data.

The combined approach of QAT and advanced quantization not only optimizes model efficiency but also enhances its generalizability, unseen data. This is particularly crucial in real-world applications where models must perform consistently under various settings and datasets, ensuring consistent and accurate results across different environments.

Typically, a bit width of 8 bits is chosen to represent both weights of model and activations in quantized models. Research indicates that employing quantization with 8-bit integer can shrink the size of model to a one fourth of its original size while boosting computational efficiency by three to four times on hardware platforms. Although reducing the bit width further can enhance computational efficiency, it often comes with the sacrifice of model accuracy (Karimzadeh and Raychowdhury, 2024). Consequently,

we've opted bit-width of 8-bit integer data for quantifying our cyber attack detection model.

Several quantization methodologies have been developed in recent literature. While retraining-based quantization is commonly utilized, it necessitates the retraining of neural network models to preserve detection accuracy, leading to increased computational expenses and time requirements (Inci et al., 2022). However, regular access of training data might not be feasible, particularly in scenarios where privacy is a concern, such as in IoT applications. Therefore, we have opted for advanced dynamic quantization for our quantization approach. Quantization aware training with Post-training dynamic quantization entails transforming weights to int8 and dynamically converting activations to int8 during computation, facilitating efficient quantization through convolution and matrix multiplication. Unlike static quantization, dynamic quantization supports gated cell structures, LSTMs, and RNNs. While it may have limitations in quantizing hidden and convolutional layers, quantization aware training and PTDQ is well-suited to our research context, offering reductions in model size while maintaining acceptable accuracy levels (Yang et al., 2022).

$$q_{\text{int}} = \text{round} \left(\frac{\text{float}}{\text{scale}} + \text{offset} \right) \quad (3.23)$$

This equation stands for the changing floating-point value (float) to an numerical value (q_{int}) by dividing the floating-point (scale) value by the and incorporating an offset to levels. The round function was utilized to validate that the result is approximated to the nearest value of integer.

$$\text{scale} = \frac{\text{float}_{\text{max}} - \text{float}_{\text{min}}}{q_{\text{max}} - q_{\text{min}}} \quad (3.24)$$

This equation calculates the scaling factor (scale) based on the min and max values of the initial floating-point parameter distribution ($\text{float}_{\text{max}}$ and $\text{float}_{\text{min}}$) and the desired precision (q_{max} and q_{min}). For 8-bit quantization, $q_{\text{max}} = 127$ and $q_{\text{min}} = -128$.

$$\text{offset} = q_{\text{min}} - \left(\frac{\text{float}_{\text{min}}}{\text{scale}} \right) \quad (3.25)$$

This equation calculates the offset (offset) to be incorporated to each level, ensuring that the minimum value of the original floating-point parameter allocation maps to the minimum quantization level (q_{min}). The offset formula determines the offset value, indicating the relative position of the floating-point zero within the quantization target range before quantization.

Where as quantization aware training will be described below: In quantization-aware training, inputs are quantized to simulate the effect of quantization during training. Below are the key equations involved: The quantization of inputs X is defined as:

$$SQ_x = Q(X) \quad (3.26)$$

where $Q(\cdot)$ denotes the quantization function, and SQ_x represents the quantized version of the inputs. The dequantization of the quantized inputs SQ_x is given by:

$$X = D(SQ_x) \quad (3.27)$$

where $D(\cdot)$ represents the dequantization function. The total loss function used in quantization-aware training is:

$$\text{Loss}_{\text{total}} = \text{Loss}_{\text{cl}} + \lambda \times \text{Loss}_{\text{qu}} \quad (3.28)$$

where Loss_{cl} denotes the classification loss, typically computed using cross-entropy loss, and Loss_{qu} represents the quantization loss. The hyperparameter λ balances the trade-off between classification accuracy and quantization performance. The quantization loss function, which measures the discrepancy between the original inputs X_i and their quantized versions $\text{round}(X_i)$, is defined as:

$$\text{Loss}_{\text{qu}} = \frac{1}{N} \sum_{i=1}^N \left| \frac{X_i - \text{round}(X_i)}{X_i} \right| \quad (3.29)$$

where N is the number of input parameters.

3.6.3 Quantized DNN-BiLSTMQ Model

This research introduces a lightweight hybrid deep learning cyber attack identification and categorization system tailored specifically for IoT environments, capitalizing on a novel fusion of a dual hidden layer Deep Neural Network (DNN) with BiLSTM architecture. While DNN exhibits remarkable proficiency in discerning intricate data patterns compared to conventional machine learning models, its streamlined network structure excels in extracting nonlinear features from network data, surpassing even the most intricate deep learning architectures in terms of simplicity and efficiency. Consequently, we have opted for DNN as the foundational structure to bolster BiLSTM's deep extraction of nonlinear feature capabilities.

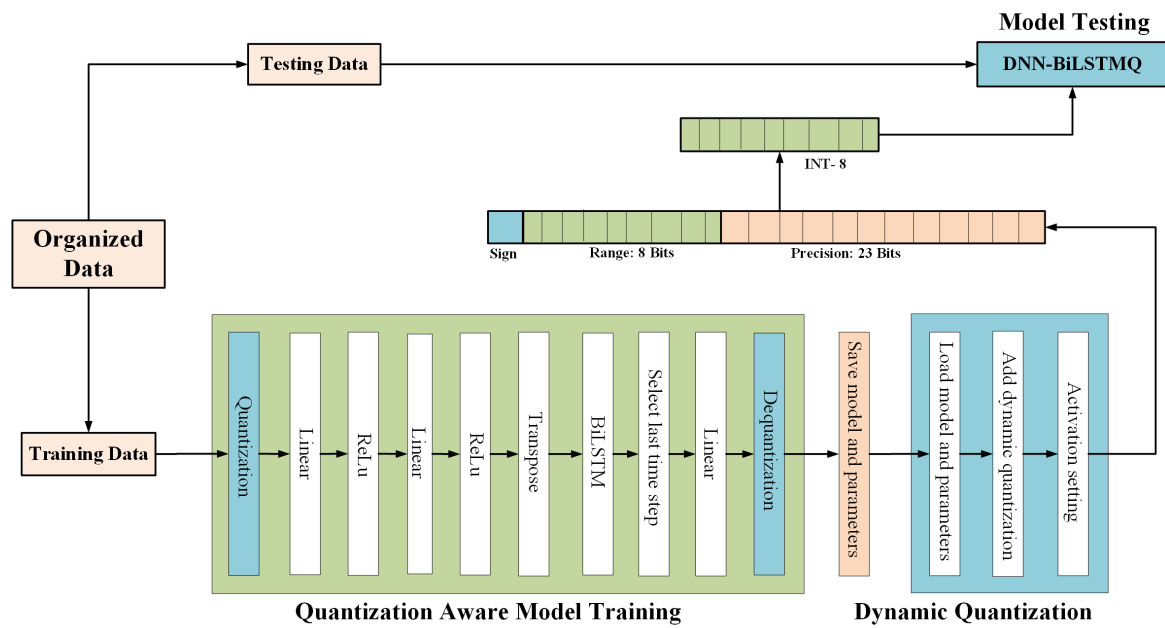
However, mixing of multiple network structures often leads to bloated model parameters, resulting in escalated computational demands. Given the critical demand for a lightweight hybrid deep learning based cyber attack detection system capable of real-time detection in resource-constrained IoT settings, it becomes paramount to strike a delicate balance between detection efficacy and computational efficiency.

In this investigation, we have implemented an advanced quantization approach, a sophisticated blend of quantization-aware training and post-training dynamic quantization, to compress the blended DNN-BiLSTM model. This innovative approach curtails the size and complicatedness of the model while preserving its feature extraction prowess and detection accuracy, thus facilitating seamless deployment in IoT environments characterized by limited resources. By leveraging techniques such as storing and loading model parameters post-training and subsequently applying a quantizer for quantization operations, we have optimized the model's efficiency without compromising its functionality. Utilizing advanced dynamic quantization will help for the generalizability of the model.

Given the constraints of current dynamic quantization operations, particularly their efficacy on structures like LSTM and Linear, we have primarily focused on quantizing the LSTM units within the BiLSTM framework. By meticulously evaluating the quantized model's performance alongside the original model parameters, we have ensured minimal accuracy loss during the phase of testing. The model architecture and development process of our proposed DNN-BiLSTM with quantization (DNN-BiLSTMQ) approach are meticulously detailed in Figure 3.4 which is adopted from (Wang et al., 2023) and modified for advanced quantization.

By implementing advanced techniques which is the combination of quantization-aware training, and post-training quantization, we have refined our cyber attack detection system to ensure optimal efficacy in IoT environments characterized by stringent resource constraints.

FIGURE 3.4: DNNBiLSTMQ Model Structure adopted and modified from (Wang et al., 2023)



3.7 Advanced Model Optimization

This study employs the optimization with Optuna hyperparameter methodology to optimize hyperparameters associated with model detection, aiming to enhance model classification accuracy. Optuna, introduced by (Akiba et al., 2019), represents a contemporary approach to hyperparameter optimization frameworks. It conceptualizes hyperparameter optimization as an iterative process aimed at maximizing or minimizing an objective function, wherein the function yields an assessment score corresponding to input hyper-parameter sets. The advancement of Optuna over conventional manual and automated tuning methodologies is predominantly evident in three key dimensions (Srinivas and Katarya, 2022).

3.7.1 Advantages of Optuna

1. **Flexible Parameter Search Space:** Optuna facilitates dynamic construction of the parameter search space, empowering users to adaptively define hyperparameter ranges for individual models. Subsequently, it generates the objective function according to interactions with the experimental entity. The subsequent search strategy is informed by historical evaluations of previous experiments, guiding the validation of forthcoming sets of hyperparameters.
2. **Efficient Optimization:** Optuna incorporates efficient optimization technique, encompassing state-of-the-art hyperparameter sampling techniques and pruning algorithms designed to terminate unpromising trails. By employing both relational & standalone sampling methodologies, Optuna swiftly identifies optimal search spaces. The pruning algorithms facilitate the premature termination of futile trails during training, a strategy commonly known as automated early halting. These optimization algorithms are particularly critical for enhancing performance in resource-constrained scenarios.
3. **Versatile and Scalable:** Optuna boasts a generic architecture and straightforward setup procedure, rendering it applicable across diverse experimental contexts. Its interactive interface facilitates seamless deployment of lightweight experiments onto large-scale distributed computing platforms, thereby meeting the demand for scalable experimentation.

3.7.2 Advanced Optuna Implementation Strategy

To apply Optuna in this research, we integrate it with DNN-BiLSTM model enhanced with quantization-aware training. The optimization process involves:

- **Dataset Preprocessing:** Convert categorical data to numerical forms suitable for model training. Features are normalized and dimensionality reduced using RAL-MIFS and two-stage IPCA techniques.
- **Objective Function Definition:** Define the parameters and their search spaces (e.g., learning rate, batch size) within the objective function. Train the optimized model with training dataset and then evaluate its efficiency on a separate testing dataset.
- **Optimization Process:** Optimize the model by iterating through a designated number of trials, refining hyperparameter values to maximize detection accuracy.

Optuna's application in hyperparameter tuning demonstrates its effectiveness in improving model detection and classification accuracy across diverse experimental domains. By leveraging its advanced optimization capabilities, this strategy enhances the overall effectiveness of the proposed detection framework.

Algorithm 2 Optuna Hyperparameter Optimization with RLA-MIFS + two stage IPCA Pseudocode

Input: The Dataset D , Number of trials, Hyperparameters and their value ranges

Output: Maximum accuracy and corresponding hyperparameters

- 1: **procedure** DATA PREPROCESSING
 - 2: Label encoding in D to one-hot encoding
 - 3: Numerically encode symbolic feature columns based on attack types
 - 4: Perform min-max normalization on features
 - 5: Feature Selection using RAL-MIFS
 - 6: $D_n \leftarrow$ Reduce feature dimensionality using two-stage IPCA
 - 7: $D_{tr}, D_{ts} \leftarrow$ Devide D_n into training & testing datasets
 - 8: **return** $D_{tr}, D_{ts} \Rightarrow k$ -dimensional feature space
 - 9: **end procedure**
 - 10: **procedure** DEFINE OBJECTIVE FUNCTION
 - 11: Define hyperparameters and their search spaces
 - 12: Train DNN-BiLSTM model on D_{tr} with quantization-aware training
 - 13: $DNN\text{-}BiLSTMQ \leftarrow$ Quantized model after training
 - 14: Test $DNN\text{-}BiLSTMQ$ on D_{ts}
 - 15: $y_{pred} \leftarrow$ Predicted values of $DNN\text{-}BiLSTMQ$
 - 16: $y_{test} \leftarrow$ True values of D_{ts}
 - 17: $A_c \leftarrow$ Accuracy of $DNN\text{-}BiLSTMQ$ on D_{ts}
 - 18: **return** $A_c \Rightarrow$ Accuracy on testing dataset
 - 19: **end procedure**
 - 20: **procedure** OPTIMIZATION PROCESS
 - 21: Define optimization objective (e.g., maximize accuracy)
 - 22: Run optimization process for a designated number of trials
 - 23: **return** Hyperparameters yielding maximum accuracy
 - 24: **end procedure**
-

Algorithm 2 outlines the Optuna hyperparameter optimization procedure, which is pivotal in enhancing the detection approach proposed in this research. The algorithm takes as input the dataset, the desired trial number, and the hyperparameters that will be optimized along with their respective search spaces. Initially, the raw dataset undergoes preprocessing to RAL-MIFS and then goes to two stage IPCA-dimensional and split data into training and test datasets.

Subsequently, the objective function for optimization is defined. For the detection approach proposed herein, the hyperparameters subject to optimization include learning rate, batch_size, and weight_decay. Accordingly, the search space of hyper-parameters is delineated within the objective function definition to facilitate subsequent hyperparameter sampling. The test efficacy of post-model training and testing serves as the objective function's return value, thereby evaluating the identification performance of different trial.

Following the objective function definition, the optimization process will be instantiated, specifying whether the goal is to maximize or minimize the outcome of objective function. In this research, the objective is to get hyperparameter values that yield the maximum test accuracy within the designated space.

Finally, the optimization commences by setting the amount of trials that will be executed. Optuna iteratively refines the hyperparameter combination within the designated search space using the hyperparameter algorithm across performed trials. The output reveals the trials exhibiting the highest identification accuracy along with their corresponding hyperparameter values. It's pertinent that the implementation encompasses RAL-MIFS + two-stage IPCA approach alongside the incorporation of quantization aware training.

Chapter 4

Experimental Results and Discussion

4.1 Experimental Setup and Evaluation Metrics

4.1.1 Experimental Setup

The assessments were meticulously conducted within the Python 3.11 environment, renowned for its versatility and robust analytical capabilities. Leveraging the PyTorch framework, known for its prowess in handling complex computations, ensured the efficiency required for sophisticated data analysis with machine learning tasks.

Execution took place on a personal computer operating on the Windows 11 platform, equipped with an 11th Gen Intel(R) Core (TM) i7-1165G7 processor. Operating at a base frequency of 2.80GHz and turbocharging up to 2701 MHz, this processor's four physical cores and eight logical processors efficiently handled the computational demands.

Additionally, the system boasted 12 GB of RAM, ensuring sufficient memory resources to accommodate intricate computational workflows without performance bottlenecks. This combination of processing power and memory capacity facilitated seamless execution and swift completion of the evaluation processes, ensuring best performance & resource usage throughout the assessment.

4.1.2 Evaluation Metrics

This research introduces an evaluation framework to assess the effectiveness of IoT attack identification models. Given the complexities in IoT attack detection, the evaluation expands beyond traditional metrics to include model size, computation time, inference time, and FLOPs.

- **Model Size:** Crucial for deployment on resource-constrained IoT devices, necessitating efficient memory use.
- **Computation Time:** Critical for real-time IoT where swift threat detection is vital.
- **Test Time:** Important for evaluating the speed of threat detection.
- **FLOPs (Floating Point Operations Per Second):** Helps understand computational overhead, guiding deployment decisions in diverse IoT environments.
- **Parameter Numbers:** Indicates the count of trainable parameters in the model, influencing its complexity and resource requirements.

The evaluation framework also includes traditional metrics:

- **Accuracy:** Measures the proportion of true results (both true positives & true negatives) among the total.
- **Precision:** Measures model's effectiveness to minimize false alarms by identifying genuine attacks.
- **Recall:** Measures model's effectiveness to correctly identify genuine attacks.
- **F1-Score:** Harmonizes precision and recall to provide a balanced evaluation.

By integrating these metrics, the research aims to comprehensively assess the model's performance, facilitating informed deployment decisions in IoT ecosystems. This approach addresses challenges in IoT attack detection, contributing to robust security solutions tailored to IoT deployment needs.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.1)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (4.2)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (4.3)$$

$$F1 - score = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (4.4)$$

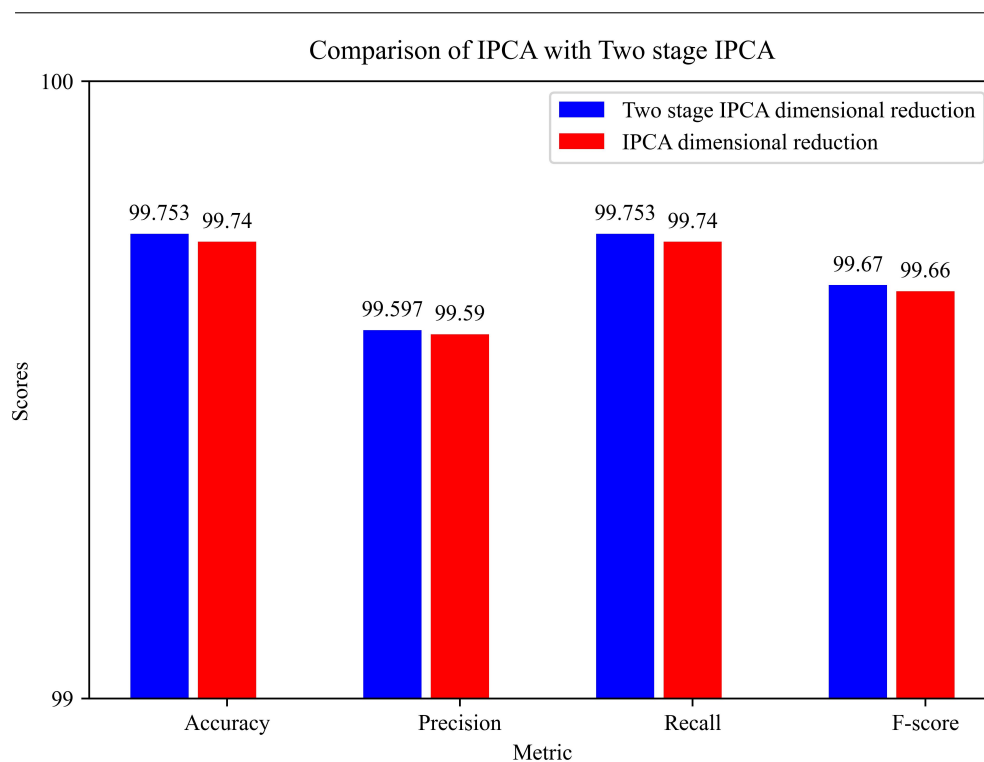
Where TP represents the true positives, indicating the count of normal traffic accurately identified as normal, and TN represents the true negatives, indicating the anomaly traffic accurately identified as anomaly traffic. FP refers to the false positives, meaning normal traffic misclassified as anomaly traffic, while FN represents the false negatives, meaning anomaly traffic misclassified as normal traffic.

4.2 Analyzing of Performance Measures on the Dataset

4.2.1 Comparision of IPCA with Two stage IPCA

This study centers on comparing and analyzing the detection accuracy achieved by IPCA, and an integration of RAL-MIFS and two-stage IPCA method as improved feature engineering techniques in cyber attack identification. The purpose is to showcase the superiority of RAL-MIFS + two-stage IPCA. By using the proposed methid we can reduce CIC IDS2017 dataset features from 78 to 26 features while maintaining an accuracy of 99.753%

FIGURE 4.1: Comparision of RAL-MIFS + IPCA/RAL-MIFS + Two stage IPCA



Experimental results shows that the RAL-MIFS with two-stage IPCA method significantly enhances the detection efficiency of cyber attacks. Specifically, by reducing CIC IDS2017 dataset features from 78 to 26, the RA-MIFS + two-stage IPCA method achieves an accuracy of 99.753%, outperforming single stage RAL-MIFS + IPCA techniques. This substantial improvement underscores the effectiveness of the RAL-MIFS + two-stage IPCA method in extracting the most vital features and thereby enhancing the performance of cyber attack detection models.

Practically speaking, the experiments validate the performance superiority of RAL-MIFS + two-stage IPCA over RAL-MIFS + IPCA, indicating its effectiveness in intrusion detection scenarios. Furthermore, this study conducts a comparison of the optimal identification efficacy of RAL-MIFS + two stage IPCA reduction algorithms using CIC IDS2017 dataset, wherein the two-stage IPCA algorithm surpasses others regarding precision, accuracy, recall, and F1-score. A comprehensive performance comparison is depicted in Figure 4.1. In summary, the RAL-MIFS + two-stage IPCA algorithm chosen in this research outperforms widely used feature engineering technique in the test and demonstrates outstanding performance.

4.2.2 Performance Measures on CIC IDS2017 Dataset

To ascertain the applicability of our proposed architecture for IoT attack recognition, this subsection undertakes a comparative analysis between the DNN-BiLSTMQ model and its constituent sub-models, namely DNN and BiLSTM, through rigorous experimentation. Furthermore, it juxtaposes our model against a spectrum of widely used ML models across several performance parameters, including accuracy, recall, precision, F1-score, training duration, and test time. The results of these comparative experiments are meticulously documented in Table 4.1.

TABLE 4.1: Comparison of multiclassification performance on CIC IDS2017 dataset.

Model	Accuracy	Precision	Recall	F1-score	Training time (s)	Test time (s)
BiLSTM	0.9967	0.9953	0.9967	0.9959	1800	19.00
2D-CNN	0.9951	0.9947	0.9951	0.9948	1350	11.00
DNN	0.9965	0.9949	0.9965	0.9956	750	9.00
DNN-BiLSTMQ	0.9973	0.9957	0.9973	0.9964	1200	10.00

From the comparison, it is evident that:

- For **accuracy**, the DNN-BiLSTMQ model achieves 99.73%, which is 0.06%, 0.22%, and 0.08% better than BiLSTM, CNN, and DNN, respectively.
- For **precision**, the DNN-BiLSTMQ model obtains 99.57%, which is 0.04%, 0.10%, and 0.08% better than BiLSTM, CNN, and DNN, respectively.
- For **recall**, the DNN-BiLSTMQ model achieves 99.73%, which is 0.06%, 0.22%, and 0.08% better than BiLSTM, CNN, and DNN, respectively.
- For the **F1-score**, the DNN-BiLSTMQ model obtains 99.64%, which is 0.05%, 0.16%, and 0.08% better than BiLSTM, CNN, and DNN, respectively.

As evidenced in Table 4.1, the DNN-BiLSTMQ model emerges as the frontrunner, demonstrating unparalleled performance across all evaluation metrics, with accuracy, F1-score, recall, and precision surpassing the 99.57% threshold. This exceptional performance is due to RAL-MIFS + two stage IPCA with the synergistic fusion of feature extraction capabilities from DNN and BiLSTM algorithms encapsulated within the DNN-BiLSTMQ framework, facilitating a more holistic feature representation.

While the BiLSTM model exhibits commendable performance in isolation, its intricate architecture necessitates substantial computational resources. Despite advanced dynamic quantization efforts aimed at model generalization and reducing model size by one-third, it is observed to compromise detection performance by small amount. However, the incorporation of the DNN component within the DNN-BiLSTMQ architecture effectively mitigates these deficiencies, endowing the model with enhanced classification prowess while maintaining computational efficiency compared to standalone local algorithms.

Noteworthy is the DNN-BiLSTMQ model's expedited test time relative to its counterparts, albeit at the sacrifice of prolonged training time of CNN and BiLSTM models. This disparity is attributed to the unified architectural design of DNN-BiLSTMQ, wherein DNN and BiLSTM are seamlessly integrated while leveraging quantization-aware training during training, which helps in model generalizability and to extract more important features. While this integration does confer substantial computational advantages during training, its efficacy becomes discernible during the test stage, particularly with the employment of dynamic quantization post-training.

Complementing these quantitative assessments, Figures 4.2, 4.3, 4.4, and 4.5 visually encapsulate the prediction outcomes of the DNN-BiLSTMQ model juxtaposed with other comparative models across diverse sample categories within CIC IDS2017 dataset, presented in confusion matrices. This graphical representation provides intuitive insights into model performance, with darker hues denoting more accurate predictions and, consequently, higher classification accuracy. Notably, the confusion matrix facilitates the discernment of false positive rates, a critical metric delineating the misclassification of negative samples as benign.

FIGURE 4.2: BiLSTM Quantitative Assessment

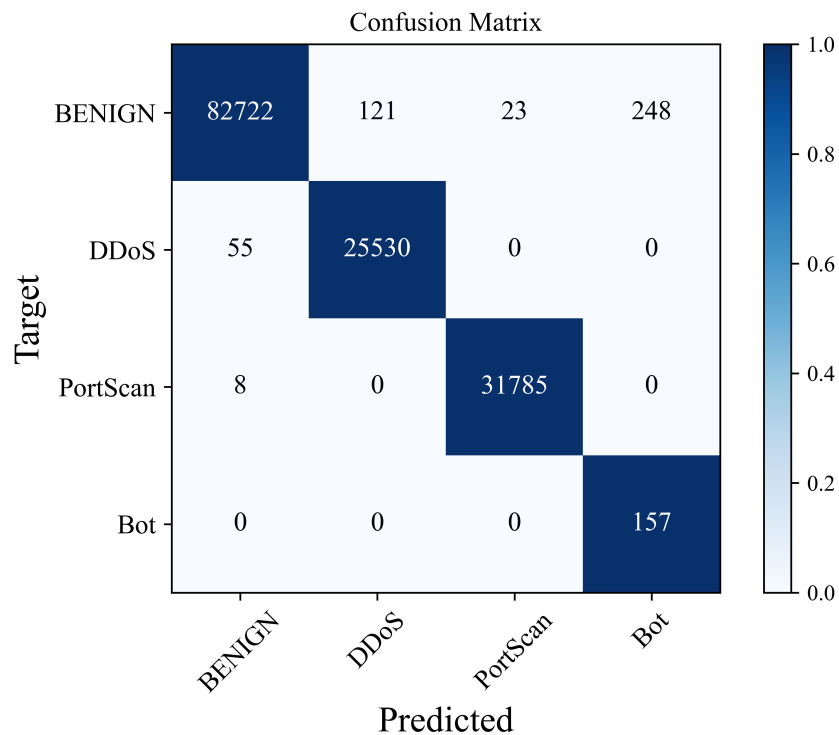


FIGURE 4.3: DNN Quantitative Assessment

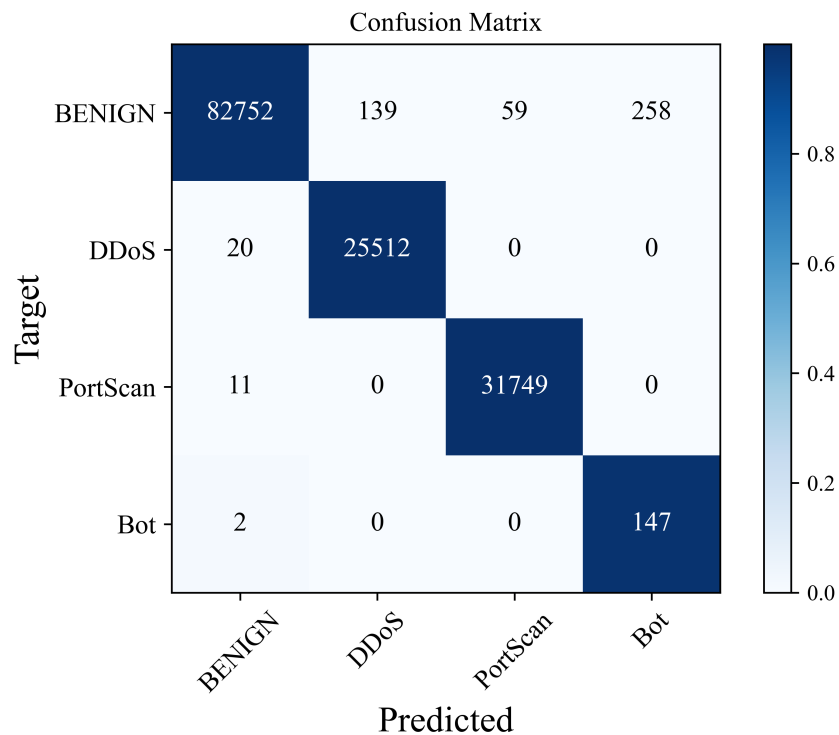


FIGURE 4.4: CNN Quantitative Assessment

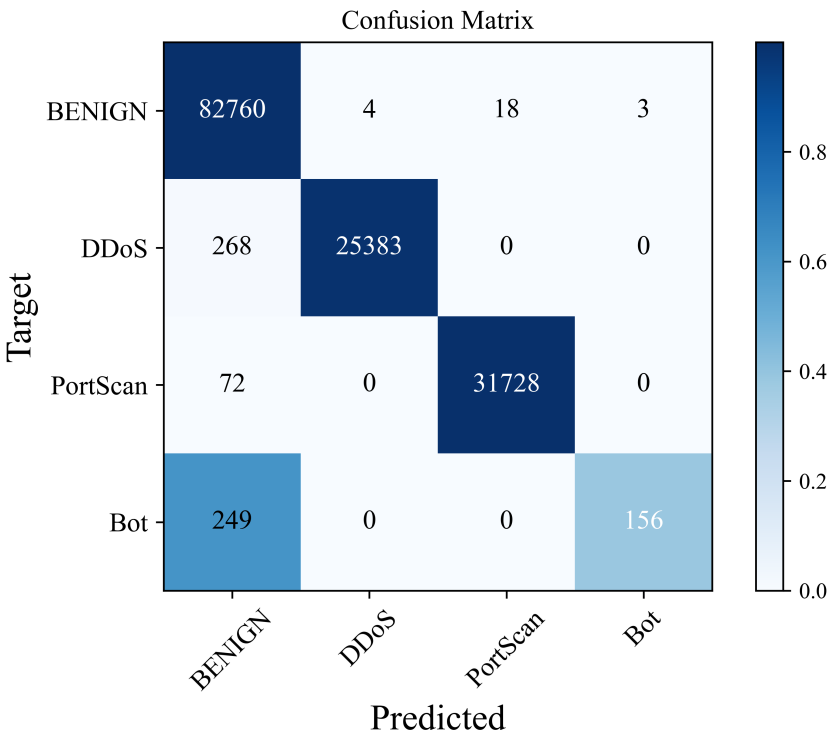
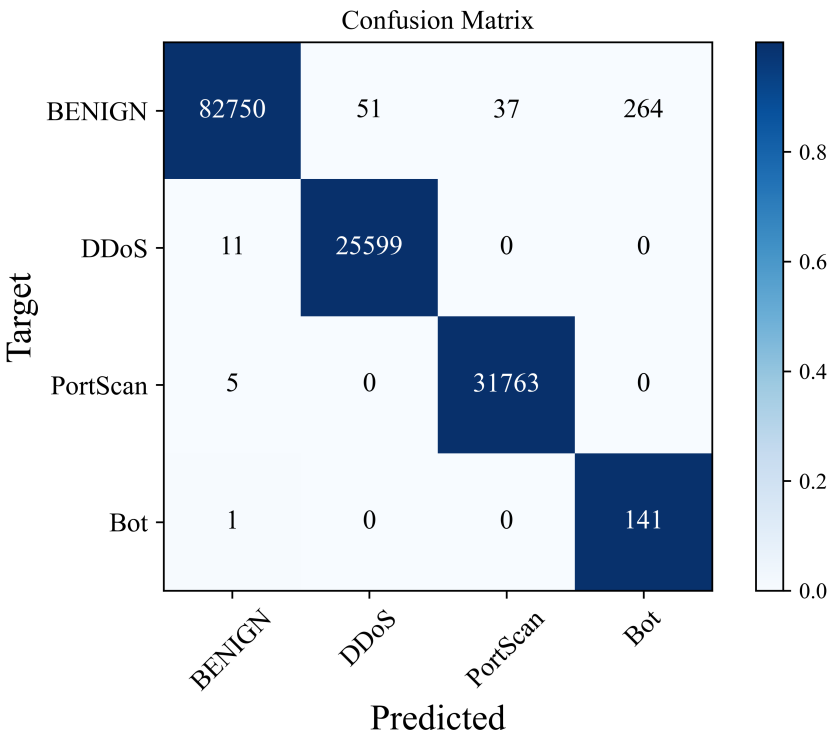


FIGURE 4.5: DNN-BiLSTMQ Quantitative Assessment



4.2.3 Performance Measure on CICIOT2023 dataset

To ascertain the efficacy of the DNN-BiLSTMQ model in detecting novel IoT cyber attacks, this study evaluates the method using CICIOT2023 dataset and compares it against other deep learning methods. The results of this comparison are presented in Table 4.2. The experimental findings substantiate that the proposed DNN-BiLSTMQ model surpasses alternative models in identifying recent attack data across all evaluation parameters. Notably, this model demonstrates shorter training and test times, particularly when contrasted with DNN models, where it exhibits a notable advantage in training duration, being merely one-third of the BiLSTM.

TABLE 4.2: Comparison of multiclassification performance on CICIOT2023 dataset.

Model	Accuracy	Precision	Recall	F1-score	Training Time (s)	Test Time (s)
BiLSTM	0.9307	0.9227	0.9303	0.9176	3450	26.20
2D-CNN	0.9296	0.9221	0.9296	0.9180	4000	27.80
DNN	0.9275	0.9194	0.9275	0.9141	1650	16.00
DNN-BiLSTMQ	0.9395	0.9305	0.9395	0.9290	3500	20.20

From the comparison, it can be evident that:

- For **accuracy**, the DNN-BiLSTMQ model achieves 93.95%, which is 0.73%, 0.84%, and 1.05% better than BiLSTM, CNN, and DNN, respectively.
- For **precision**, the DNN-BiLSTMQ model achieves 93.05%, which is 0.73%, 0.79%, and 1.06% better than BiLSTM, CNN, and DNN, respectively.
- For **recall**, the DNN-BiLSTMQ model achieves 93.95%, which is 0.73%, 0.84%, and 1.05% better than BiLSTM, CNN, and DNN, respectively.
- For the **F1-score**, the DNN-BiLSTMQ model achieves 92.90%, which is 0.94%, 0.90%, and 1.29% better than BiLSTM, CNN, and DNN, respectively.

This comprehensive assessment underscores the efficacy of the DNN-BiLSTMQ model in discerning novel IoT network attacks, further reaffirming its utility in contemporary intrusion detection scenarios.

FIGURE 4.6: BiLSTM Quantitative Assessment

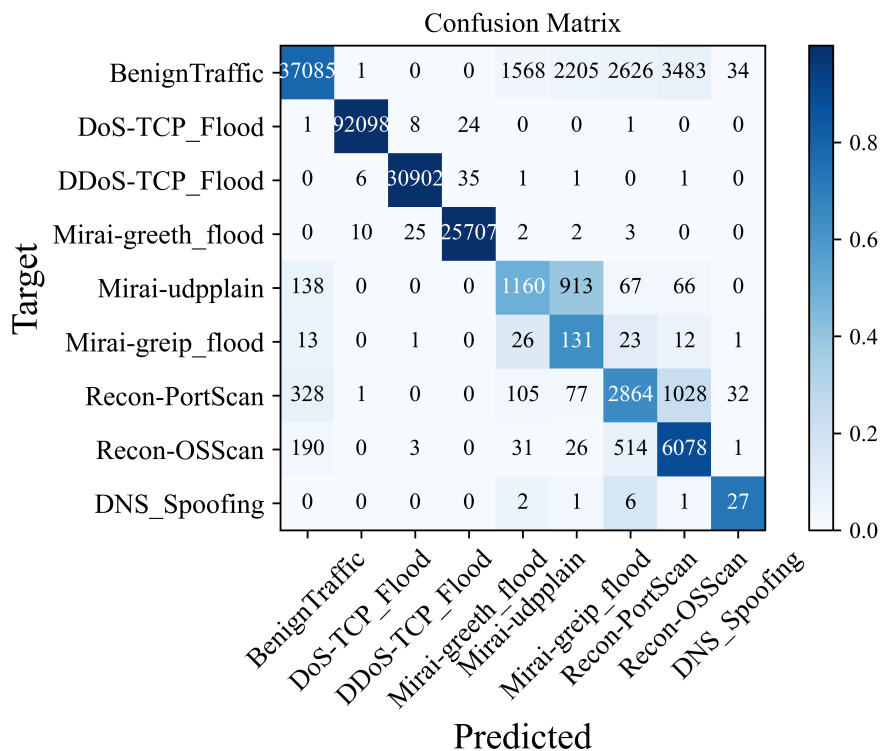


FIGURE 4.7: CNN Quantitative Assessment

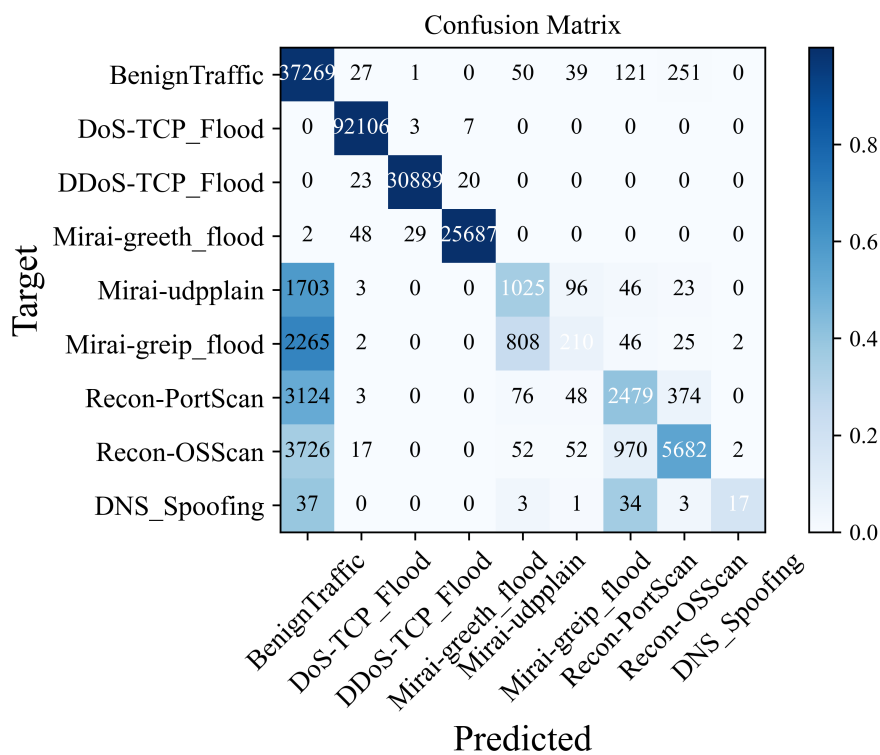


FIGURE 4.8: DNN Quantitative Assessment

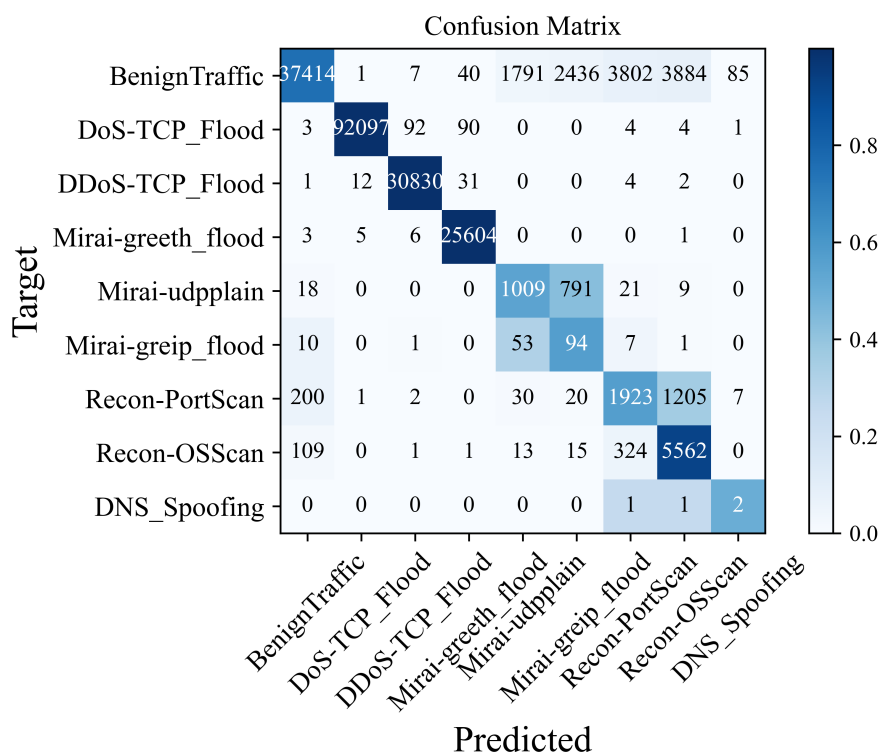
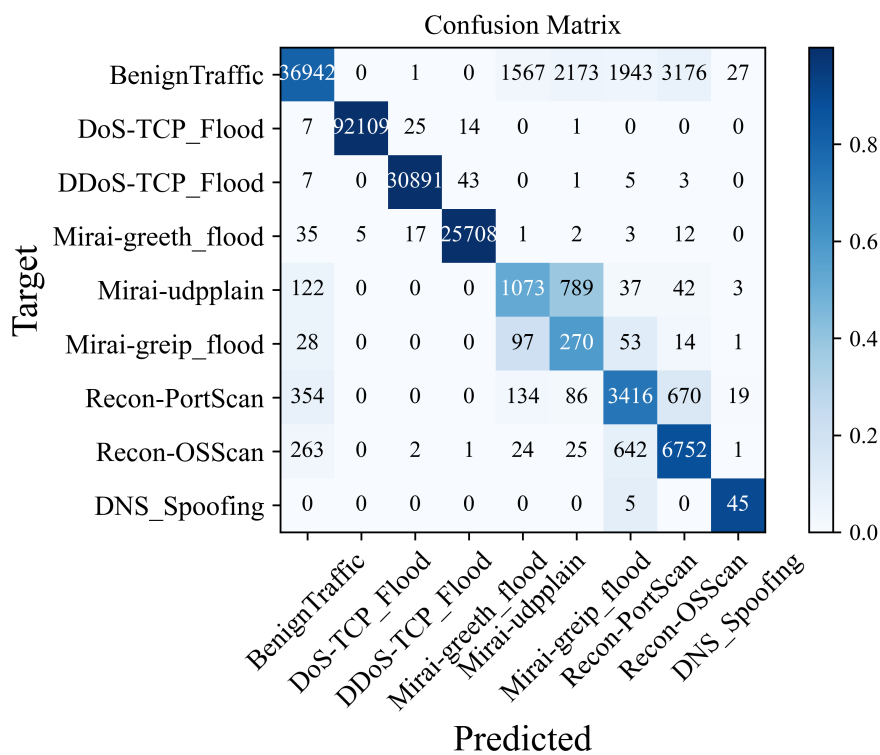


FIGURE 4.9: DNN-BiLSTMQ Quantitative Assessment



In cybersecurity, it is imperative to achieve a high F1-score. The experiments indicate that both models for both datasets have high F1-scores. The comprehensive and enhanced feature engineering method, which involved a thorough investigation of the dataset, enabled us to identify the optimal features. This led to nearly optimal performance in terms of recall, precision, and F1-score for detection machine learning models.

4.3 Analysis of Lightweight Model Performance

This section evaluates the lightweight characteristics and classification performance of the proposed model, addressing the limited processing capabilities of smart devices. The comparison encompasses various deep learning methods, focusing on model size, complexity, and floating-point operations (FLOPs). The assessment is conducted separately for the CIC IDS2017 & CICIOT2023 datasets. Detailed results are presented in Tables 4.3a and 4.3b.

TABLE 4.3: Comparison of Attack Detection Accuracy on Different Datasets

(A) CIC IDS2017 Dataset

Model	Params	FLOPs	Size (KB)
BiLSTM	18724	456768	47.56
2D-CNN	1488	558080	40.00
DNN	33244	12888960	46.46
DNN-BiLSTMQ	1000	2784	25.60

(B) CICIOT2023 Dataset

Model	Params	FLOPs	Size (KB)
BiLSTM	16522	7144,704	62.42
2D-CNN	19017	9598976	145.00
DNN	27657	11415040	44.00
DNN-BiLSTMQ	2473	7700	31.30

4.3.1 Model Lightweightness Analysis on CIC IDS2017 Dataset

DNN-BiLSTMQ model demonstrates significant lightweight characteristics:

- **Parameter Reduction:** 99.39% compared to DNN, 99.50% compared to CNN, and 99.98% compared to BiLSTM.
- **Complexity Reduction:** 94.66% compared to BiLSTM, 30.94% compared to CNN, and 96.99% compared to DNN.
- **Model Size:** The quantized model is only 46.17% of BiLSTM, 36.00% of DNN, and 44.90% of CNN.

4.3.2 Model Lightweightness Analysis on CICIoT2023 Dataset

The DNN-BiLSTMQ model maintains its lightweight efficiency:

- **Parameter Reduction:** 99.94% compared to BiLSTM, 99.92% compared to CNN, and 99.93% compared to DNN.
- **Complexity Reduction:** 91.06% compared to DNN, 87.23% compared to CNN, and 91.41% compared to BiLSTM.
- **Model Size:** 49.52% of BiLSTM, 28.86% of DNN, and 78.41% of CNN.

4.3.3 Key Factors for Lightweight Attributes

1. **Improved Feature Engineering:** The best feature was extracted from high dimensional data and then transformed into a low-dimensional space using RAL-MIFS + two-stage IPCA algorithm.
2. **Advanced Dynamic Quantization:** Advanced dynamic quantization (QAT+PTDQ) of the model structure results in a smaller size of model, suitable for deployment on IoT architecture with limited resources.

The empirical evidence underscores the suitability of the proposed DNN-BiLSTMQ model for IoT environments. The model offers lightweight yet high-performing cyber attack detection solutions, making it ideal for deployment in resource-constrained settings.

4.4 Comparing Proposed Architecture with State-of-the-Art Techniques

TABLE 4.4: Comparison of models' multiclassification effectiveness.

Ref.	Datasets	Accuracy	Precision	Recall	F1-score
(Zhao et al., 2021b)	CIC-IDS2017	99.11%	99.10%	99.00%	99.00%
	TON-IoT	98.7%	98.4%	98.4%	98.11%
(He et al., 2023)	TON-IoT	99.40%	99.37%	99.40%	99.38%
(Yao et al., 2023)	WSN-DS	97.91%	97.92%	98.54%	97.91%
	USW-NB15	98.92%	98.93%	98.02%	98.91%
	KDD CUP99	98.23%	98.23%	98.23%	98.23%
(Fatani et al., 2022)	BloT	99.41%	99.03%	99.03%	99.02%
	NSL-KDD	99.30%	99.30%	99.28%	76.11%
(Wang et al., 2023)	CIC-IDS2017	99.67%	99.54%	99.67%	99.59%
	CICIoT2023	93.13%	91.80%	93.13%	91.94%
Proposed Model	CIC IDS2017	99.73%	99.57%	99.73%	99.64%
		0.06% ↑	0.03%	0.06%	0.05%
	CICIoT2023	93.95%	93.05%	93.95%	92.90%

TABLE 4.5: Comparison of models' lightwightness.

Ref.	Datasets	Model size (KB)	FLOPS	No of params
(Wang et al., 2023)	CIC-IDS2017	28.3	1654	610800
	CICIoT2023	36.3	1988	628800
Proposed Model	CIC-IDS2017	25.6	2784	1000
		8.13% ↓	39.54%↓	99.548%↓
	CICIoT2023	31.3	7700	2473

From the Table 4.4 and 4.5 it is evident with the same dataset our method has excellent detection and classification rate in addition to reduction in model size. These results indicate the efficacy of the proposed system incoparsion to other models, with a consistent advantage of at least 0.06% in accuracy, 0.03% in precision, 0.06% in recal, 0.05% in F1-score, 8.0% redection in mdl size, 39.54% reduction in FLOPs and 99.54% reduction in No of params compared to (Wang et al., 2023).

4.5 Deployment Architecture

In edge computing (Shi and Dustdar, 2016) data processing occurs at the network edge, bringing computation closer to data sources rather than relying on cloud or fog computing. In the context of a smart home automation system, an edge device acts as a computing or networking resource positioned between home IoT devices and cloud servers, enabling faster processing and response times.

Fog computing (Idrissi, Azizi, and Moussaoui, 2022) is an emerging model positioned close to IoT devices, enhancing cloud capabilities by providing local computing, storage, and networking resources. Serving as an intermediary layer, it bridges IoT devices with cloud data centers. This model consists of distributed, diverse computational nodes, which offer localized processing to reduce latency and improve efficiency.

In IoT networks (Isong, Kgote, and Abu-Mahfouz, 2024), cyber attack detection systems are deployed using host-based and network-based architectures (Shi and Dustdar, 2016). Host-based deployment operates on IoT devices at fog nodes, monitoring device-specific activities to detect localized threats. Network-based deployment is positioned at gateways or within network infrastructure, analyzing traffic patterns to identify distributed attacks across the network.

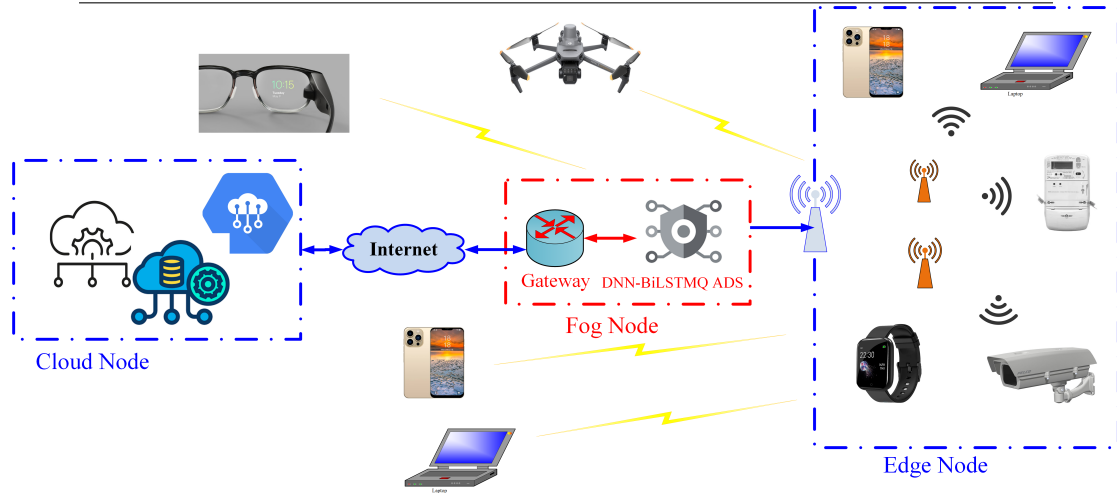
Our proposed hybrid deep learning-based host-based attack detection system (HDL-ADS) is designed for deployment on lightweight IoT devices situated at the fog node 4.10, which possess significantly less computational power than traditional servers or firewalls. In a real-time IoT environment the system is intended to be installed at the network edge on AI-enabled chips, ideally with a direct power source connection. The scenario is designed for smart home automation system

By positioning the HDL-ADS at the edge node, we can ensure faster inference regardless of network connectivity, as all processing is conducted locally on the device. This setup excludes the need for data traffic to and from a cloud server, thereby speeding up inference times and preventing any potential data leakage. The model is optimized to occupy minimal space and utilize limited memory resources, which enhances latency performance. However, this optimization necessitates a trade-off between model size and accuracy.

The HDL-ADS is specifically configured to analyze only inbound traffic, operating under the assumption of the device itself is legitimate and trustworthy. If additional security measures are required, a network-based attack detection system (NADS) should

be deployed at the fog node to bolster security. This supplementary ADS would benefit from more computational power and could be connected to a lightweight hardware accelerator designed for deep neural networks, thereby enhancing its efficacy.

FIGURE 4.10: implementation architecture



We should create and implement our HDL-ADS model using Python for the Raspberry Pi and the Arduino Integrated Development Environment (IDE) for microcontroller boards. For model inference, we will employ compatible libraries and techniques suitable for specific hardware platforms, such as microcontrollers featuring 32-bit processors based on the ARM Cortex-M Series architecture, including the Arduino, ESP32, and other similar devices.

For the host based techniques there are two ways of deploying attack detection model:

- **Direct Deployment on Hardware:** The model can be directly deployed on hardware platforms such as the Arduino Portenta H7, STM32 Nucleo Board, and Raspberry Pi 3B+/4.
- **Integration into Main Source Code:** The model can be decomposed into header (.h) and source (.c) files. These files can then be integrated into the main source code on each microcontroller, making the model part of the full program.

This dual deployment strategy showcases the model's versatility and applicability across various hardware platforms. To optimize our deployment strategy, we will deploy our model on the FoG node, positioned after the gateway. This approach facilitates real-time processing and reduces latency by bringing computation closer to the data source. Utilizing the computational resources of the FoG node ensures efficient handling of model inference, providing quick and reliable responses for the application.

Chapter 5

Conclusion

In addressing the challenges posed by large-scale, sophisticated IoT cyber attack identification and classification, besides the need for lightweight & real-time detection and classification solutions, this research proposes a hybrid DL based cyber attack identification and classification system integrating DNN & BiLSTM model.

Our proposed methodology introduces a robust RAL-MIFS + two-stage IPCA techniques for reducing feature dimension during data preprocessing. It leverages DNN-BiLSTM architectures to extract nonlinear, bidirectional long-range features from network data. Furthermore, we use advanced dynamic quantization and Optuna hyperparameter optimization to model unit structure pre and post-training, resulting in a DNN-BiLSTMQ (Quantized) model tailored for lightweight cyber attack detection.

To assess the effectiveness of our approach, we have done an experiments using CIC IDS2017 and CICIOT2023 datasets, which simulate industrial and network IoT environments. Our experimental demonstrate superior detection performance compared to existing methods across both datasets. Additionally, our approach offers smaller model sizes and lower complexities compared to alternative detection methods.

The proposed methodology in this research demonstrates commendable detection performance and exhibits a lightweight nature. The implementation of advanced dynamic quantization which integrated quantization aware training + dynamic quantization for post training quantization makes the model generalizability excellent. However, it is essential to acknowledge certain limitations. The experiment took more training time and might not fully emulate the intricacies and dynamics of IoT network environments. Therefore, it is a necessity to subject the model to rigorous testing using more intricate datasets by integrating real-time test. Furthermore, it will contribute to fortifying the robustness and reliableness of the detection and classification mechanisms developed.

Bibliography

- Abdalgawad, N. et al. (2022). "Generative Deep Learning to Detect Cyberattacks for the IoT-23 Dataset". In: *IEEE Access* 10, pp. 6430–6441. DOI: [10.1109/ACCESS.2021.3140015](https://doi.org/10.1109/ACCESS.2021.3140015).
- Abdulhammed, Razan et al. (2019). "Features Dimensionality Reduction Approaches for Machine Learning Based Network Intrusion Detection". In: *Electronics* 8.3. ISSN: 2079-9292. DOI: [10.3390/electronics8030322](https://doi.org/10.3390/electronics8030322). URL: <https://www.mdpi.com/2079-9292/8/3/322>.
- Agbedanu, P et al. (2022). "A Lightweight Intrusion Detection for Internet of Things Using Incremental Ensemble Learning". In: *Journal Not Specified* 1.0.
- Akiba, Takuya et al. (2019). "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631.
- Alalhareth, Mousa and Sung-Chul Hong (2023). "An Improved Mutual Information Feature Selection Technique for Intrusion Detection Systems in the Internet of Medical Things". In: *Sensors* 23.10, p. 4971.
- Alkahtani, Hasan and Theyazn HH Aldhyani (2021). "Intrusion detection system to advance internet of things infrastructure-based deep learning algorithms". In: *Complexity* 2021.
- Almomani, Iman and Afnan Alromi (2020). "Integrating software engineering processes in the development of efficient intrusion detection systems in wireless sensor networks". In: *Sensors* 20.5, p. 1375.
- Alshahrani, Hani Mohammed (2021). "Coll-iot: A collaborative intruder detection system for internet of things devices". In: *Electronics* 10.7, p. 848.
- Alsulami, Abdulaziz A et al. (2022). "An Intrusion Detection and Classification System for IoT Traffic with Improved Data Engineering". In: *Applied Sciences* 12.23, p. 12336.
- Alzubi, Qusay M et al. (2022). "Intrusion detection system based on hybridizing a modified binary grey wolf optimization and particle swarm optimization". In: *Expert Systems with Applications* 204, p. 117597.
- Awajan, Albara (2023). "A novel deep learning-based intrusion detection system for IOT networks". In: *Computers* 12.2, p. 34.

- Bhandari, Guru et al. (2023). "Distributed Deep Neural-Network-Based Middleware for Cyber-Attacks Detection in Smart IoT Ecosystem: A Novel Framework and Performance Evaluation Approach". In: *Electronics* 12.2, p. 298.
- Boppana, Tej Kiran and Priyanka Bagade (2023). "GAN-AE: An unsupervised intrusion detection system for MQTT networks". In: *Engineering Applications of Artificial Intelligence* 119, p. 105805.
- Cai, Hanfei et al. (2022). "Short-Term Load Forecasting Based on Radam Optimized CNN-BiLSTM-AE Hybrid Model". In: *2022 Power System and Green Energy Conference (PSGEC)*. IEEE, pp. 626–631.
- Derhab, Abdelouahid et al. (2020). "Intrusion detection system for internet of things based on temporal convolution neural network and efficient feature engineering". In: *Wireless Communications and Mobile Computing* 2020, pp. 1–16.
- Din, Ikram Ud et al. (2019). "Machine learning in the Internet of Things: Designed techniques for smart cities". In: *Future Generation Computer Systems* 100, pp. 826–843.
- Elhoseny, Mohamed et al. (2021). "Security and privacy issues in medical internet of things: overview, countermeasures, challenges and future directions". In: *Sustainability* 13.21, p. 11645.
- Fatani, Abdulaziz et al. (2022). "Advanced feature extraction and selection approach using deep learning and Aquila optimizer for IoT intrusion detection system". In: *Sensors* 22.1, p. 140.
- Gabsi, Souhir et al. (2021). "Survey: Vulnerability Analysis of Low-Cost ECC-Based RFID Protocols against Wireless and Side-Channel Attacks". In: *Sensors* 21.17, p. 5824.
- Gao, Mengnan et al. (2023). "Anomaly traffic detection in IoT security using graph neural networks". In: *Journal of Information Security and Applications* 76, p. 103532.
- Gopali, Saroj and Akbar Siami Namin (2022). "Deep Learning-Based Time-Series Analysis for Detecting Anomalies in Internet of Things". In: *Electronics* 11.19, p. 3205.
- Han, Hyojoon, Hyukho Kim, and Yangwoo Kim (2022). "An efficient hyperparameter control method for a network intrusion detection system based on proximal policy optimization". In: *Symmetry* 14.1, p. 161.
- He, Mingshu et al. (2023). "A lightweight and efficient IoT intrusion detection method based on feature grouping". In: *IEEE Internet of Things Journal*.
- Hoque, Nazrul, Dhruva K Bhattacharyya, and Jugal K Kalita (2014). "MIFS-ND: A mutual information-based feature selection method". In: *Expert systems with applications* 41.14, pp. 6371–6385.
- Ibibo, Joshua Teddy (2023). "Emerging Challenges and Solutions in RPL Protocol: Research Review". In: *2023 IEEE 28th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, pp. 283–289.

- Idrissi, Idriss, Mostafa Azizi, and Omar Moussaoui (2022). "A lightweight optimized deep learning-based host-intrusion detection system deployed on the edge for IoT". In: *International Journal of Computing and Digital System*.
- Inci, Ahmet et al. (2022). "Qadam: Quantization-aware dnn accelerator modeling for pareto-optimality". In: *arXiv preprint arXiv:2205.13045*.
- Isong, Bassey, Otshepeng Kgote, and Adnan Abu-Mahfouz (2024). "Insights into Modern Intrusion Detection Strategies for Internet of Things Ecosystems". In: *Electronics* 13.12, p. 2370.
- Karimzadeh, Foroozan and Arijit Raychowdhury (2024). "Twofold Sparsity: Joint Bit- and Network-Level Sparsity for Energy-Efficient Deep Neural Network Using RRAM Based Compute-In-Memory". In: *IEEE Access*.
- Kaur, Arvinder, Saibal K Pal, and Amrit Pal Singh (2018). "Hybridization of K-means and firefly algorithm for intrusion detection system". In: *International Journal of System Assurance Engineering and Management* 9.4, pp. 901–910.
- Khan, Muhammad Ashfaq (2021). "HCRNNIDS: Hybrid convolutional recurrent neural network-based network intrusion detection system". In: *Processes* 9.5, p. 834.
- Khater, Belal Sudqi Abed Saleh (2021). "A Lightweight Host-Based Intrusion Detection System Using N-Gram and Perceptron Model for Internet of Things". PhD thesis. University of Malaya (Malaysia).
- Kiourtis, Athanasios et al. (2018). "Exploring the complete data path for data interoperability in cyber-physical systems". In: *International Journal of High Performance Computing and Networking* 12.4, pp. 339–349.
- Kumar, Ankit et al. (2021). "Black hole attack detection in vehicular ad-hoc network using secure AODV routing algorithm". In: *Microprocessors and Microsystems* 80, p. 103352.
- Lahasan, Badr and Hussein Samma (2022). "Optimized deep autoencoder model for internet of things intruder detection". In: *IEEE Access* 10, pp. 8434–8448.
- Lin, Kunda, Xiaolong Xu, and Fu Xiao (2022). "MFFusion: A Multi-level Features Fusion Model for Malicious Traffic Detection based on Deep Learning". In: *Computer Networks* 202, p. 108658.
- Maheswaran, N et al. (2022). "Multistage intrusion detection system using machine learning algorithm". In: *Mobile Computing and Sustainable Informatics*. Springer, pp. 139–153.
- Mekruksavanich, Sakorn and Anuchit Jitpattanakul (2021). "Biometric user identification based on human activity recognition using wearable sensors: An experiment using deep learning models". In: *Electronics* 10.3, p. 308.

- Mendonca, Robson V et al. (2022). "A lightweight intelligent intrusion detection system for industrial internet of things using deep learning algorithms". In: *Expert Systems* 39.5, e12917.
- Mullen, Gary and Liam Meany (2019). "Assessment of buffer overflow based attacks on an IoT operating system". In: *2019 Global IoT Summit (GloTS)*. IEEE, pp. 1–6.
- Murugesan, G and P Maheswaravenkatesh (2024). "A Lightweight Convolutional Neural Network based network intrusion detection and classification method for Social Internet of Things". In.
- Nazir, Anjum et al. (2023). "A novel feature-selection algorithm in IoT networks for intrusion detection". In: *Sensors* 23.19, p. 8153.
- Otoutum, Yazan, Dandan Liu, and Amiya Nayak (2022). "DL-IDS: a deep learning-based intrusion detection framework for securing IoT". In: *Transactions on Emerging Telecommunications Technologies* 33.3, e3803.
- Ozawa, S, S Pang, and N Kasabov (2010). "Online feature extraction for evolving intelligent systems". In: *Evolving intelligent systems: Methodology and applications*, pp. 151–171.
- Papaioannou, Maria et al. (2022). "A survey on security threats and countermeasures in internet of medical things (IoMT)". In: *Transactions on Emerging Telecommunications Technologies* 33.6, e4049.
- Pham, Vinh, Eunil Seo, and Tai-Myoung Chung (2020). "Lightweight Convolutional Neural Network Based Intrusion Detection System." In: *J. Commun.* 15.11, pp. 808–817.
- Qaddoura, Raneem et al. (2021). "A multi-layer classification approach for intrusion detection in iot networks based on deep learning". In: *Sensors* 21.9, p. 2987.
- Rajendran, Gowthamaraj et al. (2019). "Modern security threats in the Internet of Things (IoT): Attacks and Countermeasures". In: *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, pp. 1–6.
- Rani, Shalli et al. (2022). "Lightweight Hybrid Deep Learning Architecture and Model for Security in IIOT". In: *Applied Sciences* 12.13, p. 6442.
- Rizi, Mohammad Hosein Panahi and Seyed Amin Hoseini Seno (2022). "A systematic review of technologies and solutions to improve security and privacy protection of citizens in the smart city". In: *Internet of Things*, p. 100584.
- Saba, Tanzila et al. (2022). "Anomaly-based intrusion detection system for IoT networks through deep learning model". In: *Computers and Electrical Engineering* 99, p. 107810.
- Sadiq, Ali Safa et al. (2019). "Fraud detection model based on multi-verse features extraction approach for smart city applications". In: *Smart cities cybersecurity and privacy*. Elsevier, pp. 241–251.

- Said, Rachid Ben, Zakaria Sabir, and Iman Askerzade (2023). "CNN-BiLSTM: A Hybrid Deep Learning Approach for Network Intrusion Detection System in Software Defined Networking with Hybrid Feature Selection." In: *IEEE Access*.
- Sanju, P (2023). "Enhancing intrusion detection in IoT systems: A hybrid metaheuristics-deep learning approach with ensemble of recurrent neural networks". In: *Journal of Engineering Research* 11.4, pp. 356–361.
- Sharafaldin, Iman, Arash Habibi Lashkari, Ali A Ghorbani, et al. (2018). "Toward generating a new intrusion detection dataset and intrusion traffic characterization." In: *ICISSp* 1, pp. 108–116.
- Shi, Weisong and Schahram Dustdar (2016). "The promise of edge computing". In: *Computer* 49.5, pp. 78–81.
- Srinivas, Polipireddy and Rahul Katarya (2022). "hyOPTXg: OPTUNA hyper-parameter optimization framework for predicting cardiovascular disease using XGBoost". In: *Biomedical Signal Processing and Control* 73, p. 103456.
- Sudqi Khater, Belal et al. (2019). "A lightweight perceptron-based intrusion detection system for fog computing". In: *applied sciences* 9.1, p. 178.
- Tian, Qiao, Jingmei Li, and Haibo Liu (2019). "A Method for Guaranteeing Wireless Communication Based on a Combination of Deep and Shallow Learning". In: *IEEE Access* 7, pp. 38688–38695. DOI: [10.1109/ACCESS.2019.2905754](https://doi.org/10.1109/ACCESS.2019.2905754).
- Tian-yi, Yang, Liu Shi-yue, and Liu Jun-yi (2019). "Network traffic anomaly detection based on incremental possibilistic clustering algorithm". In: *Journal of Physics: Conference Series*. Vol. 1284. 1. IOP Publishing, p. 012067.
- Ullah, Imtiaz and Qusay H Mahmoud (2022). "An Anomaly Detection Model for IoT Networks based on Flow and Flag Features using a Feed-Forward Neural Network". In: *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, pp. 363–368.
- Ullah, Safi et al. (2022). "A new intrusion detection system for the internet of things via deep convolutional neural network and feature engineering". In: *Sensors* 22.10, p. 3607.
- Wang, Ching-Hao et al. (2022). "Lightweight deep learning: an overview". In: *IEEE Consumer Electronics Magazine*.
- Wang, Zhendong et al. (2023). "A lightweight intrusion detection method for IoT based on deep learning and dynamic quantization". In: *PeerJ Computer Science* 9, e1569.
- Wang, Zhendong et al. (2024). "A lightweight IoT intrusion detection model based on improved BERT-of-Theseus". In: *Expert Systems with Applications* 238, p. 122045.

- Yang, Jianxun et al. (2022). "GQNA: Generic quantized DNN accelerator with weight-repetition-aware activation aggregating". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 69.10, pp. 4069–4082.
- Yao, Wenbin et al. (2023). "A lightweight intelligent network intrusion detection system using one-class autoencoder and ensemble learning for IoT". In: *Sensors* 23.8, p. 4141.
- Zhang, Zhibo et al. (2022). "Explainable artificial intelligence applications in cyber security: State-of-the-art in research". In: *IEEE Access*.
- Zhao, Ruijie et al. (2021a). "A novel intrusion detection method based on lightweight neural network for internet of things". In: *IEEE Internet of Things Journal* 9.12, pp. 9960–9972.
- (2021b). "A novel intrusion detection method based on lightweight neural network for internet of things". In: *IEEE Internet of Things Journal* 9.12, pp. 9960–9972.

Appendix A

Appendix

A.1 Glossary of All Terms

This section is organized to help the reader get aware of some technical terms that he/she would come across quite frequently while reading this research.

Advanced Dynamic Quantization: A technique used in machine learning to reduce the model size and increase computational efficiency. This involves dynamically quantizing the weights and activations of the neural network during inference, leading to faster execution with minimal loss in accuracy.

Analysis: Involves examining various types of attacks, including port scanning, spam, and penetration of HTML files.

Autoencoder: This type of neural network consists of an encoder and decoder that compress and then reconstruct the input data, learning the most significant features in the process.

Backdoors: Techniques used by attackers to bypass security mechanisms and gain unauthorized access to systems or data.

Bidirectional Long Short-Term Memory (BiLSTM): A type of recurrent neural network (RNN) that improves upon standard RNNs by allowing information to flow in both forward and backward directions. BiLSTMs are particularly effective for tasks involving sequential data, such as text or time-series analysis, because they can capture context from both past and future states, providing a more comprehensive understanding of the sequence.

Bot Attack: A type of cyber attack in which automated software, known as bots, is used to carry out malicious activities. These activities can include spamming, data scraping,

brute force attacks, distributed denial-of-service (DDoS) attacks, and spreading malware. Bot attacks can target websites, servers, and networks, often overwhelming them with a high volume of automated requests or compromising their security.

Classification: A supervised learning technique used when the output variable is categorical, with no inherent order among the categories.

Clustering: An unsupervised learning method that groups data based on similarity measures.

Convolutional Neural Network (CNN): A type of deep neural network particularly suited for analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing.

DDOS: Distributed Denial of Service (DDOS) refers to a type of network attack where multiple systems overwhelm a target system with traffic.

Deep Learning: A subset of machine learning within AI, deep learning uses deep neural networks with multiple hidden layers to process data. It includes models such as deep neural networks (DNN), recurrent neural networks (RNN), and long short-term memory (LSTM), and can handle unstructured data.

Deep Neural Network (DNN): A type of artificial neural network with multiple layers between the input and output layers. DNNs can model complex non-linear relationships and are capable of learning features from data at various levels of abstraction. They are versatile and can be applied to a wide range of tasks, including image and speech recognition, and natural language processing.

DNS Spoofing: A type of cyber attack in which a malicious actor corrupts the Domain Name System (DNS) cache or responses, redirecting traffic from a legitimate website to a fraudulent one. This can result in users unknowingly providing sensitive information to malicious sites or experiencing service disruptions. DNS spoofing can be used for phishing, spreading malware, or intercepting sensitive data by tricking the victim into thinking they are connecting to a trusted site.

Exploits: Attacks leveraging known system vulnerabilities to gain unauthorized access or control.

Fuzzers: Tools that generate random data to test systems for vulnerabilities by attempting to cause crashes.

Incremental Component Analysis: An approach in machine learning that updates the model incrementally as new data becomes available. This technique is particularly

useful for handling large datasets or streaming data, as it allows for continuous learning without the need to retrain the model from scratch.

Machine Learning: A branch of Artificial Intelligence (AI), machine learning involves algorithms that analyze data to make decisions.

Naïve Bayes: A supervised learning algorithm based on Bayes' theorem, often used for classification.

Neural Networks: Also known as artificial neural networks, these machine learning techniques mimic the human brain's structure, where neurons are connected by synapses, and weights simulate the strength of these connections.

Quantization Aware Training (QAT): A technique used in training neural networks that simulates the effects of quantization during the training process. Quantization involves reducing the precision of the weights and activations from floating-point to lower-bit representations (e.g., 8-bit integers), which can significantly reduce the model size and improve inference speed. By incorporating quantization effects during training, QAT helps the model to learn to be robust to the quantization process, thereby maintaining higher accuracy when deployed on hardware with limited precision.

Random Forest: This algorithm is used for both classification and regression, utilizing multiple decision trees to improve performance.

Reconnaissance: The process of gathering information about a system to understand its structure and identify potential vulnerabilities.

Regression: A supervised learning approach where the output variable is continuous and has a meaningful order (e.g., greater or less).

RNN: Recurrent Neural Networks (RNNs) are specialized neural networks that process sequences of data, using internal memory to remember previous inputs.

Shellcode: A small piece of code used as the payload during the exploitation of software vulnerabilities.

Supervised Learning: In this AI technique, algorithms learn to map input (x) to output (y) based on provided input-output pairs.

SVM: Support Vector Machine (SVM) is a supervised learning algorithm applicable to both classification and regression tasks.

Unsupervised Learning: This AI method finds hidden structures in data without labeled outputs.

Appendix B

Appendix

B.1 Source Code

Next, we provide the source code for essential aspects of our methodology, including data preprocessing, parameter optimization, and the architecture of each model utilized in our analysis. The attached code offers a detailed insight into the procedures undertaken to prepare and refine the data for analysis, optimize the parameters to enhance model performance, and outline the structure of each model employed in our study. These resources aim to facilitate transparency and reproducibility, enabling researchers to comprehend and replicate our methodologies effectively.

B.1.1 Data preprocessing

```
import os
import numpy as np
import pandas as pd
import warnings
import seaborn as sns
import torch
import torch.nn as nn
import torch.nn.functional as F
import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import accuracy_score, f1_score, recall_score,
precision_score, classification_report
from sklearn.decomposition import IncrementalPCA
from sklearn.feature_selection import mutual_info_classif
```

```
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from collections import Counter
import matplotlib.pyplot as plt
from thop import profile
from scipy.constants import alpha

sns.set(color_codes=True)

torch.manual_seed(1)
np.random.seed(1)
torch.manual_seed(1)
torch.cuda.manual_seed_all(1)

torch.backends.cudnn.benchmark = False
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.enabled = False

warnings.filterwarnings("ignore", message="TypedStorage is deprecated.*")
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings("ignore", message="invalid value encountered in divide")
warnings.filterwarnings("ignore", category=RuntimeWarning, module="numpy")

EPOCH = 40 # 40 train the training data n times
TIME_STEP = None # rnn time step
BATCH_SIZE = 148 #
INPUT_SIZE = 26 # 28 rnn input size
LR = 0.00188953 # 0.004453, 0.009 learning rate

DATA_PATH = r"dataset.csv"

class CustomDataset(Dataset):
    def __init__(self, features, labels):
        self.features = features
        self.labels = labels
```

```

def __len__(self):
    return len(self.labels)

def __getitem__(self, index):
    return self.features[index], self.labels[index]

def load_and_preprocess_data(data_path):
    df = pd.read_csv(data_path, header=0, low_memory=False)
    df = df.dropna()
    scaler = MinMaxScaler()
    df.iloc[:, :-1] = scaler.fit_transform(df.iloc[:, :-1])
    return df

def logistic_function(x):
    return 1 / (1 + np.exp(-x))

def redundancy_coefficient_mutual_information_ralmifs(X, y, k):
    mi = mutual_info_classif(X, y)
    mi_indices = np.argsort(mi)[::-1]
    selected_features = []
    redundancy_matrix = np.zeros((X.shape[1], X.shape[1]))
    redundancy_weights = np.ones(X.shape[1])

    for i in mi_indices:
        if len(selected_features) == 0:
            selected_features.append(i)
            continue

        redundancy = np.mean([redundancy_matrix[i][j] for j in selected_features])
        redundancy_weight = logistic_function(redundancy)
        if redundancy * redundancy_weight < 0.7: # Threshold
            selected_features.append(i)
        for j in range(X.shape[1]):
            redundancy_matrix[i][j] = np.corrcoef(X[:, i], X[:, j])[0, 1]

```

```

        redundancy_weights[i] *= logistic_function(redundancy_matrix[i][j])

    selected_features = sorted(selected_features, key=lambda idx: mi[idx] *
                               redundancy_weights[idx], reverse=True)
    return selected_features[:k]

def feature_engineering(df, input_size):
    X = df.iloc[:, :-1].values
    y = df[' Label'].values

    print(Counter(df[' Label']))
    # Feature selection using mutual information and RAL-MIFS to select k features
    k = 40 # Desired number of features to select
    selected_features = redundancy_coefficient_mutual_information_ralmifs(X, y, k)
    X_selected = X[:, selected_features]

    # First stage IPCA to reduce to 32 features
    ipca1 = IncrementalPCA(n_components=32)
    X_pca1 = ipca1.fit_transform(X_selected)

    # Second stage IPCA to reduce to input_size features
    ipca2 = IncrementalPCA(n_components=input_size)
    X_pca_new = ipca2.fit_transform(X_pca1)

    scaler = StandardScaler()
    X_new = scaler.fit_transform(X_pca_new)
    return X_new, y, selected_features

def split_data(X, y):
    x_train, x_test, y_train, y_test = train_test_split(X, y,
                                                         test_size=0.2, random_state=666)
    x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
                                                         test_size=0.2, random_state=666)
    return x_train, x_test, x_val, y_train, y_test, y_val

```

```
def create_dataloaders(x_train, y_train, x_val, y_val, x_test, y_test,
    batch_size):
    train_data = CustomDataset(x_train, y_train)
    test_data = CustomDataset(x_test, y_test)
    val_data = CustomDataset(x_val, y_val)

    train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
    val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=False)
    return train_loader, val_loader, test_loader

# Main execution
df = load_and_preprocess_data(DATA_PATH)
X_new, y, selected_features = feature_engineering(df, INPUT_SIZE)
print("Selected Features:", selected_features)

x_train, x_test, x_val, y_train, y_test, y_val = split_data(X_new, y)
train_loader, val_loader, test_loader = create_dataloaders(x_train, y_train,
x_val, y_val, x_test, y_test, BATCH_SIZE)
```

B.1.2 Optuna parameter optimization with RLA-MIFS

```
# Constants
EPOCH = 10
BATCH_SIZE = 128
INPUT_SIZE = 24
LR = 0.001
DATA_PATH = r"dataset.csv"

warnings.filterwarnings("ignore", message="TypedStorage is deprecated.*")
warnings.filterwarnings("ignore", category=UserWarning,
                        message="dropout option adds dropout after all but last
                                recurrent layer")
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings("ignore", message="invalid value encountered in divide")
warnings.filterwarnings("ignore", category=RuntimeWarning, module="numpy")

class CustomDataset(Dataset):
    def __init__(self, features, labels):
        self.features = features
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, index):
        return self.features[index], self.labels[index]

def load_and_preprocess_data(data_path):
    df = pd.read_csv(data_path, header=0, low_memory=False)
    df = df.dropna()
    scaler = MinMaxScaler()
    df.iloc[:, :-1] = scaler.fit_transform(df.iloc[:, :-1])
    return df
```

```

def logistic_function(x):
    return 1 / (1 + np.exp(-x))

def redundancy_coefficient_mutual_information_lrgumifs(X, y, k):
    mi = mutual_info_classif(X, y)
    mi_indices = np.argsort(mi)[::-1]
    selected_features = []
    redundancy_matrix = np.zeros((X.shape[1], X.shape[1]))
    redundancy_weights = np.ones(X.shape[1])

    for i in mi_indices:
        if len(selected_features) == 0:
            selected_features.append(i)
            continue

        redundancy = np.mean([redundancy_matrix[i][j] for j in selected_features])
        redundancy_weight = logistic_function(redundancy)
        if redundancy * redundancy_weight < 0.6: # Threshold
            selected_features.append(i)
        for j in range(X.shape[1]):
            redundancy_matrix[i][j] = np.corrcoef(X[:, i], X[:, j])[0, 1]
            redundancy_weights[i] *= logistic_function(redundancy_matrix[i][j])

    selected_features = sorted(selected_features, key=lambda idx: mi[idx] *
                                redundancy_weights[idx], reverse=True)
    return selected_features[:k]

def feature_engineering(df, input_size):
    X = df.iloc[:, :-1].values
    y = df[' Label'].values

    print(Counter(df[' Label']))
    # Feature selection using RGU-MIFS to select k features
    k = 40 # Desired number of features to select
    selected_features = redundancy_coefficient_mutual_information_lrgumifs(X, y, k)

```

```

X_selected = X[:, selected_features]

# First stage IPCA to reduce to 32 features
ipca1 = IncrementalPCA(n_components=32)
X_pca1 = ipca1.fit_transform(X_selected)

# Second stage IPCA to reduce to input_size features
ipca2 = IncrementalPCA(n_components=input_size)
X_pca_new = ipca2.fit_transform(X_pca1)

scaler = StandardScaler()
X_new = scaler.fit_transform(X_pca_new)
return X_new, y, selected_features

def split_data(X, y):
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
        random_state=666)
    x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
        test_size=0.2, random_state=666)
    return x_train, x_test, x_val, y_train, y_test, y_val

def create_dataloaders(x_train, y_train, x_val, y_val, x_test, y_test,
    batch_size):
    train_data = CustomDataset(x_train, y_train)
    test_data = CustomDataset(x_test, y_test)
    val_data = CustomDataset(x_val, y_val)

    train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
    val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=False)
    return train_loader, val_loader, test_loader

class DNNBiLSTM(nn.Module):
    def __init__(self, input_size):

```

```

    super().__init__()
    self.hidden1 = nn.Linear(input_size, 48)
    self.hidden2 = nn.Linear(48, 24)
    self.lstm = nn.LSTM(
        input_size=input_size,
        hidden_size=12,
        dropout=0.2,
        bidirectional=True,
    )
    self.fc = nn.Linear(12 * 2, 4)

    # Add QuantStub and DeQuantStub
    self.quant = QuantStub()
    self.dequant = DeQuantStub()

def forward(self, x):
    x = self.quant(x)
    x = F.relu(self.hidden1(x))
    x = F.relu(self.hidden2(x))

    # Reshape or transpose x to match LSTM input format (seq_len, batch, input_size)
    # Assuming x has shape (batch_size, input_size)
    x = x.unsqueeze(0) # Add a dummy sequence length dimension at dim 0
    r_out, (h_n, h_c) = self.lstm(x)
    r_out = r_out[-1]
    x = self.fc(r_out)
    model = self.dequant(x) # Dequantize before returning
    return model.squeeze(0) # Remove the dummy sequence length dimension

def objective(trial):
    batch_size = trial.suggest_int("batch_size", 64, 256, step=64)
    LR = trial.suggest_loguniform('lr', 10 ** (-3), 10 ** (-2))
    weight_decay = trial.suggest_loguniform('weight_decay', 10 ** (-6), 10 ** (-4))

    model = DNNBiLSTM(INPUT_SIZE).to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=LR, weight_decay

```

```

        =weight_decay)
loss_func = nn.CrossEntropyLoss()

for epoch in range(EPOCH):
    model.train()
    for step, (b_x, b_y) in enumerate(train_loader):
        b_x, b_y = b_x.float().to(device), b_y.long().to(device)
        output = model(b_x)
        loss = loss_func(output, b_y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    model.eval()
    y_predict = []
    with torch.no_grad():
        for step, (a_x, a_y) in enumerate(val_loader):
            a_x = a_x.float().to(device)
            test_output = model(a_x)
            pred_y = torch.max(test_output, 1)[1]
            y_predict.extend(pred_y.cpu().numpy().tolist())

    accuracy = accuracy_score(y_val, y_predict)
    print('Epoch: ', epoch, '| train loss: %.4f' % loss.item(), '|
        validation accuracy: %.4f' % accuracy)

return accuracy

```

```

df = load_and_preprocess_data(DATA_PATH)
X_new, y, selected_features = feature_engineering(df, INPUT_SIZE)
print("Selected Features:", selected_features)

x_train, x_test, x_val, y_train, y_test, y_val = split_data(X_new, y)
train_loader, val_loader, test_loader = create_dataloaders(x_train,
y_train, x_val, y_val, x_test, y_test, BATCH_SIZE)

```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=4)

# Print the best trial parameters
print("Best trial:")
trial = study.best_trial
print(f"  Value: {trial.value}")
print("  Params: ")
for key, value in trial.params.items():
    print(f"    {key}: {value}")

# Evaluate on the test set with the best parameters
best_batch_size = trial.params['batch_size']
best_lr = trial.params['lr']
best_weight_decay = trial.params['weight_decay']

model = DNNBiLSTM(INPUT_SIZE).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=best_lr,
weight_decay=best_weight_decay)
loss_func = nn.CrossEntropyLoss()

for epoch in range(EPOCH):
    model.train()
    for step, (b_x, b_y) in enumerate(train_loader):
        b_x, b_y = b_x.float().to(device), b_y.long().to(device)
        output = model(b_x)
        loss = loss_func(output, b_y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    model.eval()
    y_predict = []
    with torch.no_grad():
        for step, (a_x, a_y) in enumerate(test_loader):
```

```
a_x = a_x.float().to(device)
test_output = model(a_x)
pred_y = torch.max(test_output, 1)[1]
y_predict.extend(pred_y.cpu().numpy().tolist())

accuracy = accuracy_score(y_test, y_predict)
print('Epoch: ', epoch, '| train loss: %.4f' % loss.item(), '|
test accuracy: %.4f' % accuracy)

print('Final test accuracy: %.4f' % accuracy)
```

B.1.3 Models

B.1.3.1 BiLSTM

```

class BiLSTM(nn.Module):
    def __init__(self):
        super(BiLSTM, self).__init__()
        self.lstm = nn.LSTM(
            input_size=INPUT_SIZE,
            hidden_size=36,
            bidirectional=True,
        )
        self.fc = nn.Linear(36 * 2, 4)

        # QAT related layers
        self.quant = torch.quantization.QuantStub()
        self.dequant = torch.quantization.DeQuantStub()

    def forward(self, x):
        x = self.quant(x) # Quantize the input
        x = x.transpose(0, 1) # input : [max_len, batch_size, n_class]
        r_out, _ = self.lstm(x)
        r_out = r_out[-1] # [batch_size, n_hidden * 2]
        model = self.fc(r_out) # model : [batch_size, n_class]
        x = self.dequant(x) # Dequantize the output
        return model

bilstm = BiLSTM().to(device=device)
bilstm.qconfig = torch.quantization.get_default_qat_qconfig('fbgemm')
torch.quantization.prepare_qat(bilstm, inplace=True)

# Dynamic quantization before testing
bilstm.eval()
quantized_bilstm = torch.quantization.convert(bilstm)
quantized_model = torch.quantization.quantize_dynamic(
    quantized_bilstm, {nn.LSTM, nn.Linear}, dtype=torch.qint8
)

```

B.1.3.2 CNN

```

class CNN(nn.Module):
    def __init__(self):

        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.bn1 = nn.BatchNorm2d(32)
        self.relu = nn.ReLU(inplace=True)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=2)
        self.bn2 = nn.BatchNorm2d(64)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=1)
        self.bn3 = nn.BatchNorm2d(128)

        self.fc = nn.Linear(128, 4)
        self.quant = QuantStub()
        self.dequant = DeQuantStub()

    def forward(self, x):
        x = self.quant(x)
        x = self.relu(self.bn1(self.conv1(x)))
        x = self.relu(self.bn2(self.conv2(x)))
        x = self.relu(self.bn3(self.conv3(x)))
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        x = self.dequant(x)
        return x

# Prepare the model for QAT
bilstm = CNN().to(device=device)
bilstm.qconfig = torch.quantization.get_default_qat_qconfig('fbgemm')
torch.quantization.prepare_qat(bilstm, inplace=True)

# Dynamic quantization before testing
bilstm.eval()
quantized_bilstm = torch.quantization.convert(bilstm)

```

```
quantized_model = torch.quantization.quantize_dynamic(  
    quantized_bilstm, {nn.Conv2d, nn.BatchNorm2d, nn.ReLU, nn.Linear},  
    dtype=torch.qint8  
)
```

B.1.3.3 DNN

```
class DNN(nn.Module):
    def __init__(self, input_size=26): # Input size
        super(DNN, self).__init__()
        self.hidden1 = nn.Linear(input_size, 120)
        self.hidden2 = nn.Linear(120, 240)
        self.out = nn.Linear(240, 4)

        self.quant = torch.quantization.QuantStub()
        self.dequant = torch.quantization.DeQuantStub()

    def forward(self, x):
        x = self.quant(x) # Quantize the input
        x = F.relu(self.hidden1(x))
        x = F.relu(self.hidden2(x))
        x = self.out(x)
        x = self.dequant(x) # Dequantize the output
        return x

DNNM = DNN(input_size=26).to(device=device)
DNNM.qconfig = torch.quantization.get_default_qat_qconfig('fbgemm')
torch.quantization.prepare_qat(DNNM, inplace=True)

# Dynamic quantization before testing
DNNM.eval()
quantized_DNN = torch.quantization.convert(DNNM)
quantized_DNNM = torch.quantization.quantize_dynamic(
    quantized_DNN, {nn.Linear}, dtype=torch.qint8
)
```

B.1.3.4 DNN-BiLSTMQ

```

# noinspection PyUnusedLocal
class DNN(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.hidden1 = nn.Linear(input_size, 24) # 32 First hidden layer
        self.hidden2 = nn.Linear(24, 12) # 32 16 Second hidden layer
        self.lstm = nn.LSTM(
            input_size=12, # 16 Adjusted to match the new size of the hidden layer
            hidden_size=6, # 8 Reduced hidden size
            dropout=0.3,
            bidirectional=True, # Bidirectional LSTM
        )
        self.fc = nn.Linear(6 * 2, 4) # 8 * 2 Output layer

        # Quantization-aware training (QAT) related layers
        self.quant = torch.quantization.QuantStub()
        self.dequant = torch.quantization.DeQuantStub()

    def forward(self, x):
        x = self.quant(x) # Quantize the input
        x = F.relu(self.hidden1(x)) # First fully connected-ReLU activation
        x = F.relu(self.hidden2(x)) # Second fully connected- ReLU activation
        x = x.transpose(0, 1) # Transpose for LSTM input
        r_out, (h_n, h_c) = self.lstm(x) # LSTM layer
        r_out = r_out[-1] # Select the output of the last time step
        x = self.fc(r_out) # Fully connected output layer
        x = self.dequant(x) # Dequantize the output
        return x

# Create model instance
bilstm = DNN(INPUT_SIZE).to(device=device)
bilstm.qconfig = torch.quantization.get_default_qat_qconfig('fbgemm')
torch.quantization.prepare_qat(bilstm, inplace=True)

# # Print the original model

```

```
# print("Original Model:")
# print(bilstm)

optimizer = torch.optim.Adam(bilstm.parameters(), lr=LR)
loss_func = nn.CrossEntropyLoss()

total = sum([param.nelement() for param in bilstm.parameters()])
print("Number of parameter: %.4fM" % (total / 1e6))

# Training with QAT
for epoch in range(EPOCH):
    print(epoch)
    epoch_start_time = time.time()
    train_acc = 0.0
    train_loss = 0.0
    val_acc = 0.0
    val_loss = 0.0

    bilstm.train()
    for step, (b_x, b_y) in enumerate(train_loader):
        b_x = b_x.view(-1, 1, INPUT_SIZE).float().to(device)
        b_y = b_y.long().to(device) # Convert b_y to Long type
        train_pred = bilstm(b_x) # model output
        loss = loss_func(train_pred, b_y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_acc += np.sum(np.argmax(train_pred.cpu().data.numpy(), axis=1)
                             == b_y.cpu().numpy())
        train_loss += loss.item()

    bilstm.eval()
    with torch.no_grad():
        for step, (a_x, a_y) in enumerate(val_loader):
            a_x = a_x.view(-1, 1, INPUT_SIZE).float().to(device)
            a_y = a_y.long().to(device) # Convert a_y to Long type
```

```

        val_pred = bilstm(a_x)
        loss = loss_func(val_pred, a_y)

        val_acc += np.sum(np.argmax(val_pred.cpu().data.numpy(), axis=1)
                           == a_y.cpu().numpy())
        val_loss += loss.item()
    print('[%03d/%03d] %2.2f sec(s) Train Acc: %3.6f Loss: %3.6f |
          Val Acc: %3.6f Loss: %3.6f' % \
          (epoch + 1, EPOCH, time.time() - epoch_start_time,
           train_acc / len(train_loader.dataset), train_loss /
                               len(train_loader.dataset),
           val_acc / len(val_loader.dataset), val_loss /
                               len(val_loader.dataset)))

def print_size_of_model(model):
    torch.save(model.state_dict(), "dnnbilstmQtemp.p")
    print('Size (MB):', os.path.getsize("dnnbilstmQtemp.p") / 1e6)
    os.remove('dnnbilstmQtemp.p')

print("Original Model Size:")
print_size_of_model(bilstm)

# Convert the model to quantized version after training
bilstmC = torch.quantization.convert(bilstm, inplace=True)
bilstmC.eval()

# Post-training dynamic quantization for specified layers
quantized_dnnbilstm = torch.quantization.quantize_dynamic(
    bilstmC, {nn.Linear}, dtype=torch.qint8
)

print("Quantized Model Size:")
print_size_of_model(quantized_dnnbilstm)

from thop import profile

# Create an instance of the model

```

```

dnn_model = DNN(INPUT_SIZE).to(device)

input = torch.randn(1, 6, INPUT_SIZE).to(device)
macs, params = profile(dnn_model, inputs=(input,))
print("FLOPs and params during QAT:")
print('FLOPS QAT: ', macs * 2)
print('params QAT: ', params)

# Saving the quantized model
torch.save(quantized_dnnbilstm, 'dnnbilstmQ_V5.pth')
torch.save(quantized_dnnbilstm.state_dict(), 'dnnbilstmQ_V5_feature.pth')
def get_model_flops_and_params(model, input_size):
    # Initialize FLOPs count
    flops = 0

    # Calculate FLOPs for Linear layers
    flops_hidden1 = 2 * input_size * 24
    flops_hidden2 = 2 * 24 * 12

    # LSTM: Assuming simplified calculation
    lstm_input_size = 12
    lstm_hidden_size = 6
    lstm_output_size = 12
    flops_lstm = 2*(lstm_input_size*lstm_hidden_size + lstm_hidden_size ** 2) * 4

    # Output linear layer
    flops_fc = 2 * lstm_output_size * 4

    # Total FLOPs
    total_flops = flops_hidden1 + flops_hidden2 + flops_lstm + flops_fc

    # Calculate total parameters
    paramsq = sum(p.numel() for p in model.parameters())

    return total_flops, paramsq

```

```
ptdq_flops, ptdq_params = get_model_flops_and_params(
    quantized_dnnbilstm, INPUT_SIZE)

print(f"PTDQ Model - FLOPs: {ptdq_flops}, Params: {ptdq_params}")

test_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.ToTensor(),
])

def confusion_matrix(preds, labels, conf_matrix):
    preds = torch.argmax(preds, 1)
    for p, t in zip(preds, labels):
        conf_matrix[p, t] += 1
    return conf_matrix

from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score, cla

warnings.filterwarnings("ignore")

# Loading the quantized model
basis = torch.load("dnnbilstmQ_V5.pth")
basis.load_state_dict(torch.load('dnnbilstmQ_V5_feature.pth'))
basis.eval()

prediction_basis = []
accuracy_list = []
precision_list = []
recall_list = []
f1_list = []
test_acc = 0.0
precision = 0.0
conf_matrix = torch.zeros(4, 4)
start = time.time()
```

```

with torch.no_grad():
    for step, (c_x, c_y) in enumerate(test_loader):
        c_x = c_x.view(-1, 1, INPUT_SIZE).float()
        test_pred = basis(c_x)
        test_acc += np.sum(np.argmax(test_pred.cpu().data.numpy(),
axis=1) == c_y.numpy())
        acc = accuracy_score(c_y, np.argmax(test_pred.cpu().data.numpy(),
axis=1))
        accuracy_list.append(acc)
    pre_score = precision_score(c_y, np.argmax(test_pred.cpu().data.numpy(),
axis=1), average='weighted')
    precision_list.append(pre_score)
    re_score = recall_score(c_y, np.argmax(test_pred.cpu().data.numpy(),
axis=1), average='weighted')
    recall_list.append(re_score)
    f_score = f1_score(c_y, np.argmax(test_pred.cpu().data.numpy(),
axis=1), average='weighted')
    f1_list.append(f_score)
    conf_matrix = confusion_matrix(test_pred, c_y.long(), conf_matrix)
    corrects = conf_matrix.diagonal(offset=0)
    per_kinds = conf_matrix.sum(axis=1)

labels = ['BENIGN', 'DDoS', 'PortScan', 'Bot']
plt.figure(figsize=(4, 4)) # Set a larger figure size to prevent cropping
plt.imshow(conf_matrix, cmap=plt.cm.Blues)
thresh = conf_matrix.max() / 2
for x in range(4):
    for y in range(4):
        info = int(conf_matrix[y, x])
        plt.text(x, y, info,
                verticalalignment='center',
                horizontalalignment='center',
                color="white" if info > thresh else "black")

plt.grid(False)
plt.tight_layout()

```

```
plt.yticks(range(4), labels, fontsize=12)
plt.xticks(range(4), labels, fontsize=12, rotation=45)

plt.title('Confusion Matrix')
plt.ylabel('Target', fontsize=16)
plt.xlabel('Predicted', fontsize=16)

# Adjust layout to prevent cropping
plt.tight_layout()

plt.savefig('dnn_bilstmQ_V5.png', dpi=600)
plt.show()
```