DEEP LEARNING-BASED REAL-TIME BOTNET DETECTION FOR EDGE DEVICES

Lucas Chagas de Brito Guimarães

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Rodrigo de Souza Couto

Rio de Janeiro
Novembro de 2023

DEEP LEARNING-BASED REAL-TIME BOTNET DETECTION FOR EDGE DEVICES

Lucas Chagas de Brito Guimarães

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2023

Chagas de Brito Guimarães, Lucas

Deep Learning-based Real-Time Botnet Detection for Edge Devices/Lucas Chagas de Brito Guimarães. – Rio de Janeiro: UFRJ/COPPE, 2023.

XIII, 42 p.: il.; 29, 7cm.

Orientador: Rodrigo de Souza Couto

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2023.

Referências Bibliográficas: p. 38 – 42.

1. deep learning. 2. botnet. 3. threat detection. I. de Souza Couto, Rodrigo. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

À minha família e amigos.

Agradecimentos

Agradeço imensamente à minha família por sempre incentivar e suportar meus estudos e minha formação acadêmica.

Agradeço aos professores e colegas do Grupo de Teleinformática e Automação (GTA), pelo suporte e por proporcionarem um ambiente onde pude me dedicar ao desenvolvimento deste projeto. Em particular agradeço ao meu orientador, professor Rodrigo de Souza Couto, por todo o aprendizado e apoio oferecidos; e ao professor Otto Carlos Muniz Bandeira Duarte, por me introduzir ao GTA e me incentivar a continuar meus estudos com o mestrado.

Agradeço aos professores Miguel Elias Mitre Campista e Célio Vinicius Neves de Albuquerque por aceitarem participar da banca de avaliação deste projeto.

Agradeço ao CNPq, CAPES, FAPERJ e FAPESP pelo financiamento oferecido no desenvolvimento deste trabalho. Por fim, agradeço à UFRJ, por proporcionar as oportunidades para que eu crescesse como pessoa, tanto academicamente e quanto profissionalmente. Este projeto é uma pequena forma de retribuir o investimento e a confiança em mim depositados.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

DETECÇÃO DE BOTNETS EM TEMPO REAL BASEADA EM APRENDIZADO PROFUNDO PARA DISPOSITIVOS DE BORDA

Lucas Chagas de Brito Guimarães

Novembro/2023

Orientador: Rodrigo de Souza Couto

Programa: Engenharia Elétrica

Os dispositivos da Internet das Coisas (*Internet of Things* - IoT) são fundamentais para setores como indústria 4.0, assim como casas, cidades e redes inteligentes. Apesar dos benefícios trazidos pela IoT, a existência de bilhões de dispositivos com recursos computacionais limitados os torna alvos ideais para *botnets*. Assim, várias propostas foram feitas para detectar esse tipo de ataque. No entanto, comparar diferentes propostas é difícil, uma vez que aplicam variados métodos de pré-processamento, usam diferentes algoritmos e hiperparâmetros e consideram métricas de avaliação distintas. Este trabalho implementa e compara o desempenho de oito arquiteturas de rede neural aplicadas aos conjuntos de dados BoT-IoT e N-BaIoT. A acurácia, precisão e sensibilidade dos modelos são medidas, bem como a perda durante o treinamento. Posteriormente, a vazão dos modelos em um ambiente de borda é avaliada usando um dispositivo de borda típico, o NVIDIA Jetson Nano; também é implementada a quantização pós-treinamento, avaliando-se seu impacto no desempenho dos modelos. Adicionalmente, este trabalho propõe e implementa o DL-SAFE, um IDS baseado em aprendizado profundo para detecção de *botnet* em tempo real em dispositivos de borda. Esta ferramenta implementa modelos de classificação baseados no conjunto de dados BoT-IoT em um cenário real; adicionalmente, resultados de acurácia, precisão, e sensibilidade são obtidos para avaliar o desempenho da ferramenta, demonstrando sua efetividade na detecção de ataques de negação de serviço. Os resultados demonstram que 7 dos 8 modelos avaliados apresentam precisão e sensibilidade superiores a 98%, enquanto os testes de vazão demonstram que a maioria dos modelos desenvolvidos apresentam capacidade de processamento capaz de lidar com os requisitos de rede em um ambiente típico de IoT.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DEEP LEARNING-BASED REAL-TIME BOTNET DETECTION
FOR EDGE DEVICES

Lucas Chagas de Brito Guimarães

November/2023

Advisor: Rodrigo de Souza Couto

Department: Electrical Engineering

Internet of Things devices (IoT) are fundamental for sectors such as Industry 4.0, as well as smart homes, cities, and grids. Despite the benefits brought by IoT, the existence of billions of devices with limited computing resources makes them ideal targets for botnets. Thus, several proposals have been made to detect this type of attack. However, comparing different proposals is difficult since they apply varied preprocessing methods, use different algorithms and hyperparameters, and consider distinct evaluation metrics. This work implements and compares the performance of eight neural network architectures applied to the BoT-IoT and N-BaIoT datasets. We measure the accuracy, the precision, and the recall for each model, as well as the loss during model training. Subsequently, the throughput of the models in an edge environment is evaluated using a typical edge device, the NVIDIA Jetson Nano; post-training quantization is also implemented, and its impact on model performance is evaluated. Additionally, this work proposes and implements DL-SAFE, a deep learning-based IDS for real-time botnet detection on edge devices. This tool implements classification models based on the BoT-IoT dataset in a real scenario; additionally, accuracy, precision, and recall results are obtained to evaluate the tool's performance, demonstrating its effectiveness in detecting denial of service attacks. The results demonstrate that 7 of the 8 evaluated models present precision and recall greater than 98%, while throughput tests demonstrate that most models are capable of dealing with the network requirements in a typical IoT environment.

ACRONYMS

BLSTM - Bidirectional Long Short-Term Memory

BRNN - Bidirectional Recurrent Neural Network

CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico

COPPE - Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa em Engenharia

CSV - Comma-separated Values

C&C - Command and Control

DDoS - Distributed Denial of Service

DL-SAFE - Deep Learning-based SAFeguard for Edge botnet detection

DNN - Deep Neural Network

DoS - Denial of Service

FAPERJ - Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro

FAPESP - Fundação de Amparo à Pesquisa do Estado de São Paulo

FNN - Feedforward Neural Network

GPU - Graphics Processing Unit

HTTP - Hypertext Transfer Protocol

IDS - Intrusion Detection System

IoT - Internet of Things

IP - Internet Protocol

LSTM - Long Short-Term Memory

MAC - Media Access Control

MLP - Multilayer Perceptron

OS - Operating system

PTQ - Post-Training Quantization

RAM - Random Access Memory

RNN - Recurrent Neural Network

SSL - Secure Sockets Layer

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

UFRJ - Universidade Federal do Rio de Janeiro

VM - Virtual Machine

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet of Things (IoT) is a paradigm characterized by a growing number of simple interconnected devices that can be managed remotely, usually equipped with lightweight processors [1]. IoT has become increasingly popular, being adopted in sectors such as energy, water, transportation, health, and housing, among others. According to IoT Analytics, 29 billion IoT devices will be connected to the Internet by 2027 [2].

Despite the numerous use cases, the IoT paradigm has brought challenges regarding device security. IoT devices often have vulnerabilities such as inadequate authentication, unnecessarily open ports, and inadequate access control [3]. The large number of vulnerable devices has resulted in the emergence of botnets capable of carrying out powerful Distributed Denial of Service (DDoS) attacks, such as those carried out against Yandex and Microsoft in 2021 [4, 5], and against Google in 2022 [6]. A botnet is a network of compromised computers remotely controlled by a botmaster via a Command and Control (C&C) server [7]. Botnets can take advantage of vulnerabilities in IoT devices and networks to infect machines and propagate themselves; infected machines can then be used to execute network attacks.

Simple security practices can be adopted to secure IoT devices. Such practices include changing default passwords to strong ones and keeping devices up-to-date [8]. However, these measures only act as a first line of defense. Advanced solutions must be adopted to protect devices against sophisticated attacks and identify when a device is compromised. Intrusion Detection Systems (IDSs) are an example, which can be host-based or network-based. Host-based systems run directly on each device, relying on log analysis to identify suspicious activity. Network-based systems are deployed at strategic points in the network to analyze the flow of transmitted data. As IoT devices often have limited memory and processing resources, the implementation of host-based IDSs is generally avoided, as it can interfere with device performance; thus, IDSs for IoT environments tend to be network-based [9].

Network-based IDSs can use classification models obtained from machine learning algorithms to reduce both the need for human intervention and the time required to identify attacks. This process starts by selecting a labeled dataset that contains both regular traffic data and data from the attacks that need to be detected. In addition to this selection, the algorithms' hyperparameters must also be tuned to optimize their performance. Tuning hyperparameters is usually an expensive and time-consuming process. Initially, the neural network's architecture must be defined, consisting of the number, type, and order of the layers, as well as the number of neurons per layer. In addition to the architecture, hyperparameters such as the number of epochs, the learning rate, and the batch size must also be considered. Despite the considerable time spent in the tuning process, differences in the preprocessing methods and the employed datasets make it difficult to compare models proposed by different papers.

1.1 Challenges and contributions

Due to the high financial losses commonly caused by cyber attacks [10], threat detection is currently a prominent research topic. New proposals and studies are constantly made to deal with the ever-evolving threats and to take advantage of new technologies. However, one major challenge regarding this research topic is the various methodologies employed by different authors; the lack of standardized datasets, the use of multiple preprocessing techniques, and the selection of varied hyperparameters present significant hurdles in effectively comparing and benchmarking different proposals. Moreover, although IoT devices are at greater risk of being infected by botnets, few proposals attempt to offer solutions that work directly at the edge, primarily due to the inherent limitations imposed by such devices. Relatively new products such as the NVIDIA Jetson allow GPU utilization on the edge, presenting an opportunity to apply machine and deep learning methods closer to vulnerable devices.

Aiming to assist in the construction of future models, this work implements and evaluates the performance of multiple architectures of neural networks used to detect botnet attacks in IoT networks. Two datasets are selected to build classification models, both containing labeled IoT network traffic with botnet attacks: the BoT-IoT [1] and N-BaIoT [9] datasets. A similar preprocessing method is applied to both datasets, and the same predefined set of hyperparameters is used during hyperparameter optimization. Then, the throughput of the models that exhibit the best performance for each architecture is also evaluated. Additionally, as a method of improving model throughput, quantization is applied to convert the model's weights from their original 32-bit floating points to 8-bit integers; this allows faster processing speeds at the cost of slightly lower classification accuracy. The results show that, after optimization, seven of the eight models based on the BoT-IoT dataset can classify the dataset with accuracy rates greater than 99%. Models based on the N-BaIoT dataset, on the other hand, offer lower accuracy, with a single model surpassing 85% accuracy. Through throughput tests, we also observe that, although each model's performance varies according to the implemented architecture, most models are capable of supporting the average network usage of an IoT device, which varies from 10 to 3,000 packets per second [11]. We also propose and implement DL-SAFE (Deep Learning-based SAFeguard for Edge botnet detection), an IDS for real-time traffic analysis and classification in edge environments. The BoT-IoT dataset is selected to build the tool's classification models, given that its models presented a better performance in the evaluated metrics. In our proposed tool, Open Argus[1] is used to convert, in real-time, network traffic into flows. The Pandas[2] library extracts relevant features from the converted network flows. The PyTorch framework performs flow classification. Furthermore, our tool allows for building and testing neural network architectures using three types of layers: multilayer perceptron (MLP), recurrent neural network (RNN), and long short-term memory (LSTM). The training implements hyperparameter tuning using the grid search method, and 8-bit post-training quantization (PTQ) is available as an option to improve model throughput. DL-SAFE's results show that the tool can identify at least two types of distributed denial of service (DDoS) attacks with greater than 98% accuracy.

This work's main contributions can be summarized as follows:

- **Evaluation of Multiple Neural Network Architectures**: This work implements and evaluates various neural network architectures for the detection of botnet attacks in IoT networks using two botnet datasets. This evaluation provides insights into multiple models, given that differences in the employed dataset and set of hyperparameters may lead to a substantial impact on model performance. These insights will hopefully allow for more informed choices in hyperparameter optimization for future botnet detection systems.

- **DL-SAFE Implementation**: This work introduces and implements DL-SAFE, an Intrusion Detection System (IDS) for real-time traffic analysis and classification in edge environments. The tool leverages various open-source technologies, and its code and documentation can be accessed on Github[3]. DL-SAFE has two main purposes: to allow the construction and testing of neural network architectures and to achieve accurate results in real-time IoT traffic

[1]https://openargus.org/
[2]https://pandas.pydata.org/
[3]https://github.com/GTA-UFRJ-team/neuralnetwork-IoT

classification. Additionally, it demonstrates the adaptability of its models to handle varying IoT network usage, essential for real-world deployment.

Additionally, part of this work also resulted in the following publication:

- **Guimarães, L. C. B.**, Couto, R. S. - "DL-SAFE: Proteção Baseada em Aprendizado Profundo para Detecção de Botnets na Borda", in Salão de Ferramentas do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg2023), Juiz de Fora, MG, Brazil, September 2022.

1.2 Outline

The remainder of this work is organized as follows. Section 2 presents related works that either attempt to detect botnets using deep learning methods, or propose real-time botnet detection tools. Section 3 describes and presents the features of the two botnet datasets used by this work: the BoT-IoT dataset and the N-BaIoT dataset. Section 4 presents an overview of neural networks, quickly describes the relevant types of neural network layers employed by this work, and describes the architectures used for the experiments. Section 5 describes the initial experiments and presents the models' performance results, both before and after hyperparameter optimization. Section 6 introduces DL-SAFE, describes its architecture, and presents its performance results. Finally, Section 7 concludes the work and presents future work.

Chapter 2

Related Work

Detecting botnet attacks using machine learning techniques is an issue that several authors have already worked on. Analyzing the works of these authors it is possible to study the different methods used during their respective research, including data processing and preprocessing methods, employed performance evaluation metrics, and chosen datasets. In addition, although multiple papers and surveys evaluate the classification performance of models on both newly created and previously available datasets, few recent proposals implement tools that perform this classification in real-time.

Current solutions for threat detection, including botnet detection, often rely on classic machine learning techniques [12–14]. These works have provided valuable insights into the classification performance of models considering a wide array of machine learning methods, like the decision tree, random forest, support vector machine, and naive Bayes, among others. However, comparatively few works propose solutions focusing on deep learning approaches; with advances in GPU technology making the execution of neural networks viable in lower-cost devices, it becomes essential to evaluate how deep neural networks perform when applied to security tasks such as identifying malicious network traffic. Therefore, this chapter introduces papers focusing on botnet detection through deep learning methods, as well as papers proposing tools for real-time botnet detection.

2.1 Botnet detection using deep learning methods

A work that focuses on botnet detection using deep learning is Ferrag *et al.*'s survey of deep learning-based intrusion detection systems [15]. The survey considers seven deep learning algorithms and employs them to obtain binary and multiclass classification models for the BoT-IoT and CSE-CIC-IDS2018 [16] datasets. The survey evaluates the performance of a simple architecture for each selected neural network algorithm, optimizing hyperparameters such as the number of neurons in the hidden layers and the learning rate. Ferrag *et al.* show that the IDSs reach up to 98.22% accuracy for the Deep Neural Network (DNN) model and 98.31% accuracy for the Recurrent Neural Network (RNN) model while maintaining a false positive rate below 1.15%.

Ferrag and Maglaras propose DeepCoin, an energy exchange framework for smart grids based on blockchain and deep learning [17]. The proposed framework includes a deep learning-based scheme that uses an RNN to detect network attacks. The paper tests the proposed IDS using three datasets: BoT-IoT, CICIDS2017 [18] and a Power System dataset [19]. Their proposal reaches up to 99.91% accuracy in the BoT-IoT dataset, maintaining a false positive rate of 1.28%.

Alkandi *et al.* propose a blockchain-based collaborative IDS framework, designed to preserve privacy in a cloud environment while making data exchange a simple and secure process [20]. Attacks are detected using a classification model obtained from a Bidirectional Long Short-Term Memory (BLSTM) and are evaluated using the UNSW-NB15 [21] and BoT-IoT datasets. The model achieves 98.91% accuracy and a false positive rate of less than 1%.

Popoola *et al.* propose a federated deep learning method for zero-day botnet attack detection, focusing on protecting the privacy and security of network traffic data in IoT devices [22]. The authors evaluate multiple architectures by stacking an increasing number of fully connected layers and testing different values of hidden neurons for each layer. The architectures are tested using the BoT-IoT and N-BaIoT [9] datasets. The classification model that presents the best performance for both datasets is built by stacking four fully connected layers with 100 neurons in each one, obtaining 97.04% average recall and 97.88% average recall for the BoT-IoT and N-BaIoT datasets, respectively.

Popoola *et al.* also propose SMOTE-DRNN, a deep learning algorithm for botnet detection in IoT, with a focus on handling highly imbalanced data [23]. The proposed architecture uses an RNN; the model shows high performance on the BoT-IoT dataset, achieving 99.50% precision, 99.75% recall, and 99.62% F1-score.

Saurabh *et al.* propose LBDMIDS, a network-based IDS that uses two types of LSTM to train predictive models [24]. The paper evaluates the models using the UNSW-NB15 and BoT-IoT datasets and proposes two architectures: one based on multiple LSTM layers, and another using a single BLSTM layer. Their evaluation presents an accuracy of 99.99% for both architectures.

Sualihah *et al.* propose an IDS designed to detect attacks in IoT environments [25]. The paper proposes a DNN built by stacking several fully connected layers with a variable number of neurons, as well as an LSTM-based architecture. These models are tested solely on the BoT-IoT dataset, and achieve an accuracy of 99.7% for the DNN model and 99.8% for the LSTM model.

Unlike previous works, this work initially implements and evaluates the performance of multiple neural network architectures by applying a similar preprocessing method to both datasets and using the same set of hyperparameters. Evaluating the performance of various models created with the same datasets using specific hyperparameters allows us to study each hyperparameter's influence, which is useful for future proposals dealing with similar problems or datasets. Thus, we train classification models based on the BoT-IoT and N-BaIoT datasets and have each model's performance evaluated using the same train-test subsets and metrics such as accuracy, precision, recall, and F1-Score. The throughput of the optimized models of each architecture, as well as their quantized implementations, are also evaluated to verify their applicability in IoT environments, which are more vulnerable to botnet attacks.

2.2 Real-time botnet detection

Real-time botnet detection involves the quick identification of botnet activities within a network in real-life scenarios, allowing for a fast response in cases where a botnet is detected. Intrusion Detection Systems (IDSs) play a crucial role in this process by monitoring network traffic and identifying possible botnet activity. Achieving effective real-time botnet detection demands proposals capable of handling the dynamic nature of network traffic while maintaining low latency to promptly respond to emerging threats. It is also essential to take advantage of recent advances in areas such as machine learning and edge computing; employing new technologies is essential to keep ahead of ever-evolving cyber threats such as botnets.

While several commercial projects aim to offer security against botnet attacks, such as SolarWinds' Security Event Manager[1] and ManageEngine's NetFlow Analyser[2], little research is done to provide alternative tools to deter the botnet threat. Among these proposals, Shao *et al.* poses a strategy to detect botnets using online adaptive learning and online ensemble learning [26]. Training is implemented using 2 algorithms: the adaptive Hoeffding tree and the adaptive random forest. As adaptive training is employed, a central aspect of the work is minimizing the impact of concept drift on model performance. Concept drift happens when the statistical properties of a target variable change as time passes; in this instance, it occurs due to changes in IoT network traffic patterns over time.

Velasco-Mata *et al.* perform botnet detection using machine learning on high-speed networks [27]. The authors employ a decision tree and a set of four simple

[1]https://www.solarwinds.com/security-event-manager/use-cases/botnet-detection
[2]https://www.manageengine.com/products/netflow/

features coupled with a one-second time window, aiming to optimize the proposal's performance. There is a focus on identifying the hardware requirements for the proposal to work in environments with various network requirements.

Yan *et al.* propose PeerClean, a system to detect P2P botnets in real-time [28]. Their proposal uses flow statistics and network connection behavior, in addition to a Support Vector Machine (SVM) classifier, to detect possible infected machines. PeerClean evaluated traffic from three P2P botnets, Sality, Kelihos, and ZeroAccess, respectively achieving 95.8%, 97.9%, and 100% of accuracy.

Ghafir *et al.* propose BotDet, a system for real-time detection of botnet Command and Control (C&C) traffic [29]. The proposed system operates in two stages; in the first stage, the system uses four detection modules to identify possible botnet C&C communications. These modules identify known malicious IP addresses, malicious SSL certificates, algorithmically generated domain flux [30], and connections to a Tor network. In the second stage, a correlation framework is used to reduce the false positive rate of the detection modules used during the first stage. The proposal is evaluated using third-party PCAPs, where it achieves a detection rate of 82.3% and a false positive rate of 13.6%.

As seen in the evaluated works, few works include proposals that run directly at the edge of the network. Although edge devices may have limited processing capabilities, proximity to affected devices is essential to mitigate botnet attacks before significant damage is caused. Thus, another contribution of this work is the proposal and implementation of DL-SAFE: Deep Learning-based SAFeguard for Edge botnet detection, an IDS for real-time traffic analysis and classification in edge environments. In addition to performing flow classification, the tool also allows the construction and testing of neural network architectures. The results demonstrate the tool's effectiveness in detecting DDoS attacks.

Chapter 3

Botnet Datasets

This work employs two datasets containing botnet traffic to create and evaluate classification models: the BoT-IoT and N-BaIoT datasets. This chapter describes these datasets, focusing on how they are built, their features, and the attack classes present in each.

3.1 BoT-IoT

One of the datasets selected for the analyses carried out in this work is the BoT-IoT dataset. The dataset, proposed in 2019, is created by simulating an IoT environment using virtual machines (VMs). A testbed composed of five simulated IoT devices, built using the Node-red[1] tool, is made to represent a smart home. These IoT devices are a weather station, a smart fridge, motion-activated lights, a remotely activated garage door, and a smart thermostat. Kali Linux is used to execute attacks, while the Ostinato[2] tool is used to simulate network traffic between devices. The dataset was chosen as it is relatively recent and has been used by multiple papers with a focus on botnet detection[31–34].

As the total data collected exceeds 72 million records, the dataset's authors selected a 5% subset of the data to facilitate data analysis, as well as model training and testing. As such, the BoT-IoT dataset offers both the original data with all collected records, as well as a subset containing approximately three million records. Both sets have 29 features extracted using Open Argus, while the subset has 14 additional features later extracted through data analysis techniques. The original 29 features are listed on Table 3.1, while the additional features are listed on Table 3.2. The authors also made available a reduced version of the subset containing only the 10 most relevant features, obtained after an analysis of the entropy and correlation scores of each feature.

[1]https://nodered.org/
[2]https://ostinato.org/

Table 3.1: Open Argus features used by BoT-IoT.

Feature	Description
pkSeqID	Row identifier
stime	Record start time
flgs	Flow state flags seen in transactions
flgs_number	Numerical representation of feature *flgs*
proto	Textual representation of protocols present in network flow
proto_number	Numerical representation of feature *proto*
saddr	Source IP address
sport	Source port number
daddr	Destination IP address
dport	Destination port number
pkts	Total number of packets in transaction
bytes	Total number of bytes in transaction
state	Transaction state
state_number	Numerical representation of feature *state*
ltime	Record last timestamp
seq	Argus sequence number
dur	Record total duration
mean	Average duration of aggregated records
stddev	Standard deviation of aggregated records
sum	Total duration of aggregated records
min	Minimum duration of aggregated records
max	Maximum duration of aggregated records
spkts	Source-to-destination packet count
dpkts	Destination-to-source packet count
sbytes	Source-to-destination byte count
dbytes	Destination-to-source byte count
rate	Total packets per second in transaction
srate	Source-to-destination packets per second
drate	Destination-to-source packets per second

In addition to these features, the dataset also contains three labels in order to identify whether the recorded traffic is legitimate or malicious, and in the latter's case also identify the attack's category and subcategory. The four attack categories are: scanning, theft, denial of service (DoS), and distributed denial of service (DDoS). These categories are then divided further into 10 subcategories: OS Fingerprinting, Service Scan, Keylogging, Data Exfiltration, DoS-TCP, DoS-UDP, DoS-HTTP, DDoS-TCP, DDoS-UDP, and DDoS-HTTP.

Table 3.2: Additional features employed by BoT-IoT, extracted based on Argus data.

Feature	Description
TnBPSrcIP	Total number of bytes per source IP
TnBPDstIP	Total number of bytes per destination IP
TnP_PSrcIP	Total number of packets per source IP
TnP_PDstIP	Total number of packets per destination IP
TnP_PerProto	Total number of packets per protocol
TnP_Per_Dport	Total number of packets per dport
AR_P_Proto_P_SrcIP	Average rate per protocol per source IP
AR_P_Proto_P_DstIP	Average rate per protocol per destination IP
N_IN_Conn_P_SrcIP	Total inbound connections per source IP
N_IN_Conn_P_DstIP	Total inbound connections per destination IP
AR_P_Proto_P_Sport	Average rate per protocol per sport
AR_P_Proto_P_Dport	Average rate per protocol per dport
Pkts_P_State_P_Protocol _P_DestIP	Number of packets grouped by state and protocol per destination IP
Pkts_P_State_P_Protocol _P_SrcIP	Number of packets grouped by state and protocol per source IP

The authors also evaluate the reliability of the proposed dataset using several machine learning techniques, such as Support Vector Machine, LSTM, and RNN. The models obtain, respectively, an accuracy of 100%, 97.9%, and 98.1% for the evaluated methods when using all 43 features.

3.2 N-BaIoT

Another commonly employed botnet dataset used in this work is the N-BaIoT dataset[22, 35, 36]. Unlike BoT-IoT, which uses Kali Linux and other tools to simulate botnet attacks, N-BaIoT uses real network traffic data provided by nine commercial IoT devices infected with two of the most common IoT-based botnets: Mirai and BASHLITE. The authors employ five different types of IoT devices, those being doorbells, thermostats, baby monitors, security cameras, and webcams.

The dataset, made available in 2018, has over 7 million records. The extracted features are based on 23 central features obtained for five distinct time windows, resulting in a total of 115 features; these features are presented in Table 3.3. As seen in the Table, the network flows are aggregated using four distinct methods, which are: aggregation by the same source MAC and IP addresses, aggregation by the same source IP, aggregation for the same source and destination IP addresses (channel), and aggregation for the same source and destination IP addresses and port numbers (socket).

Table 3.3: Features used by the N-BaIoT dataset.

Feature	Description
MI_dir_weight	Packet count aggregated by MAC and IP
MI_dir_mean	Mean outbound packet size aggregated by MAC and IP
MI_dir_variance	Outbound packet size variance aggregated by MAC and IP
H_weight	Packet count aggregated by source IP
H_mean	Mean outbound packet size aggregated by source IP
H_variance	Outbound packet size variance aggregated by source IP
HH_weight	Packet count aggregated by channel
HH_mean	Mean outbound packet size aggregated by channel
HH_std	Outbound packet size variance aggregated by channel
HH_magnitude	Root squared sum of the flows' packet size means aggregated by channel
HH_radius	Root squared sum of the flows' packet size variance aggregated by channel
HH_covariance	Covariance of the flows' packet size aggregated by channel
HH_pcc	Pearson correlation coefficient of the flows' packet size aggregated by channel
HH_jit_weight	Packet count aggregated by channel
HH_jit_mean	Mean time between packet arrivals
HH_jit_variance	Time variance between packet arrivals
HpHp_weight	Packet count aggregated by socket
HpHp_mean	Mean outbound packet size aggregated by socket
HpHp_std	Outbound packet size variance aggregated by socket
HpHp_magnitude	Root squared sum of the flows' packet size means aggregated by socket
HpHp_radius	Root squared sum of the flows' packet size variance aggregated by socket
HpHp_covariance	Covariance of the flows' packet size aggregated by socket
HpHp_pcc	Pearson correlation coefficient of the flows' packet size aggregated by socket

The dataset is provided as several CSV files where each filename acts as the label of the file's contents. In addition to regular traffic the dataset contains five attack classes for each botnet, totaling 10 attack classes. The BASHLITE attacks are Service Scan, Junk (sending spam data), UDP flooding, TCP flooding, and COMBO (sending spam data and opening a connection to a specific IP and port). The Mirai attacks are Service Scan, ACK Flooding, SYN Flooding, UDP Flooding, and UDPplain Flooding (optimized UDP Flooding aiming for higher packets per second).

In their work, the authors' primary objective was to classify data as legitimate or malicious through anomaly detection techniques, and their experiments achieved a true positive rate of 100% and a false positive rate of 0.7%.

Chapter 4

Neural Networks and Architectures

This chapter briefly introduces the concept of neural networks, describes some of the layer types that can be used when building neural network architectures, and presents the architectures evaluated during the tests.

4.1 Neural Networks

Neural networks typically operate by transmitting and processing data through interconnected artificial neurons organized in layers. Each neuron in a layer receives input signals, processes them based on an activation function, and produces an output signal. These activation functions are non-linear, making neural networks proficient at modeling non-linear data. These layers are categorized into three types: input layer, output layer, and hidden layer. The input layer receives the initial data to be processed, the output layer produces the final results or predictions, and the hidden layers, if present, perform additional computations [37].

The neural network's architecture is defined by its sequence of layers; when building the network, it is necessary to define each layer's type and number of neurons, and it is possible to change hyperparameters such as its activation function. Different layer types excel at different tasks; for instance, convolutional neural networks excel at performing image recognition. This work considers three types of layers when building neural network architectures: dense, Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM).

4.1.1 Dense layer and Multilayer Perceptron (MLP)

In a dense layer, each of its neurons is connected to every neuron of the preceding layer. It is one of the fundamental building blocks of neural networks and a crucial element of the multilayer perceptron.

The MLP is a neural network designed for supervised learning tasks. It is also classified as a Feedforward Neural Network (FNN), which means that information travels from the input layer to the output layer without forming any loops or cycles. MLP's training process involves two main steps, known as forward propagation and backward propagation. During forward propagation, each neuron in a layer receives inputs from the previous layer, processes them using the activation function, and generates an output. This process repeats from the input layer up to the output layer, resulting in the network's prediction. This prediction is then compared to the actual output, and their difference is quantified using a loss function. During backward propagation, this value is propagated backward through the network; this is done by calculating the derivative of the loss function, taking into account the network's weights. The weights are then updated to minimize the loss, using algorithms such as the stochastic gradient descent. This process continues over multiple epochs, improving the network's ability to make accurate predictions.

A significant issue with MLP is the vanishing or exploding gradient problems, which negatively impact model performance when using deep networks. These problems occur since, during training, each of the weights of the neural network is updated according to its gradient. The vanishing gradient problem occurs when the gradient has an extremely small value, effectively preventing the weight from changing in future iterations, while the exploding gradient problem occurs when the gradient value is too big, resulting in exponential growth.

Softmax function

An activation function commonly used on dense layers is the softmax function. This function is primarily used in the final stage of a neural network to produce output probabilities for multiclass classification tasks. The layer's function is to transform the raw output of the preceding layer into a probability distribution over multiple classes. It gives higher probabilities to higher-valued inputs while ensuring that the output values range between 0 and 1. By calculating the probability distribution for all classes, the function not only identifies the most likely class but also gives a confidence value associated with each class. This can be used to obtain insights about how confident the model is in each of its predictions.

4.1.2 Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) are a specialized type of neural network designed to process sequential or time-dependent data. Unlike the MLP and other FNNs, RNNs are known for their ability to take into account temporal dependencies through cyclic connections within the network.

RNN's main difference compared to the MLP is its ability to retain information from past iterations. In an RNN, each neuron's output is not only transmitted forward to the next layer but also sent back as an additional input, creating a cycle in the network. This recurrent feedback allows RNNs to retain information learned from previous iterations. Compared to MLPs, RNNs excel in tasks that involve sequences or patterns where the current output is also influenced by preceding inputs. This capacity to retain information over time makes RNNs particularly proficient at modeling sequential data.

Bidirectional Recurrent Neural Networks (BRNNs) are an extension of the traditional RNN architecture, aiming to take advantage of the sequential data commonly associated with RNNs. Each BRNN is composed of two RNNs, which receive the same input data in opposite directions. As the data is organized sequentially, the first entry of one RNN is the last entry of the other RNN, and vice versa; the results of the two RNNs are then concatenated to compose the BRNN result. The advantage of BRNNs is their capability to consider context from both directions, offering a more comprehensive understanding of sequence data. This is beneficial in tasks where a complete context is essential, like natural language processing.

Similar to MLP, RNNs also suffer the vanishing and exploding gradient problems when implementing deep architectures. Variations of the RNN were proposed as a way to address this problem, such as the Gated Recurrent Unit and the Long Short-Term Memory [38].

4.1.3 Long Short-Term Memory (LSTM)

LSTM networks are a specialized type of RNN designed to address the vanishing or exploding gradient problems. LSTMs mitigate these issues by introducing additional gates that allow them to selectively retain information. Each LSTM unit contains four essential components: a cell, an input gate, a forget gate, and an output gate. The cell remembers values over arbitrary time intervals, the input gate controls the information to be added to the cell, the forget gate decides what information to discard from the cell, and the output gate filters the information from the cell to produce the output.

LSTMs can also be combined with the bidirectional architecture used by BRNNS, resulting in Bidirectional LSTMs (BLSTMs). Similar to BRNNs, BLSTMs process the input sequence in both forward and backward directions, with each direction utilizing its own set of memory cells. This analysis of the sequence from both directions allows BLSTMs to better understand the data's context, incorporating information from both past and future time steps.

4.2 Architectures

The architectures selected for experiments in this work are based on the proposals presented in Chapter 2, which can be classified according to the type of layer used as the input layer. Three possible initial layers can be observed: dense, RNN, or LSTM. All architectures are presented in Table 4.1; dense layers are named Softmax if they implement the softmax activation function, or MLP when another function is used. The classification considers the 10 attack classes of the BoT-IoT and N-BaIoT datasets, presented in Chapter 3, plus a normal traffic class, totaling 11 classes for each dataset.

Table 4.1: Neural network architectures and their source papers.

Name	Architecture	Source
MLP1	MLP (100) → MLP (100) → MLP (100) → Softmax	[15]
MLP2	MLP (100) → MLP (100) → MLP (100) → MLP (100) → Softmax	[22]
RNN1	RNN (60) → Softmax	[17]
RNN2	RNN (100) → Softmax	[15]
RNND	RNN (100) → MLP (100) → MLP (100) → MLP (100) → Softmax	[23]
LSTM1	LSTM (32) → LSTM (32) → Softmax	[24]
LSTMD	LSTM (128) → LSTM (128) → MLP (32) → MLP (10) → Softmax	[25]
BLSTM1	BLSTM (12) → Softmax	[24]

The number of neurons used in each layer is shown between parentheses in Table 4.1, while the number of epochs and the batch size used during training are presented in Table 4.2.

Table 4.2: Default batch size and number of epochs used for each evaluated architecture.

Name	Batch Size	Epochs
MLP1	1000	100
MLP2	128	5
RNN1	100	5
RNN2	1000	100
RNND	64	10
LSTM1	32	5
LSTMD	128	100
BLSTM1	32	5

Two architectures composed of multiple dense layers in sequence with 100 neurons per layer are employed, with the first stacking three dense layers and the second stacking four dense layers. As both implement the MLP architecture, these are labeled MLP1 and MLP2. Three architectures starting with an RNN layer are also considered. The first two are short architectures, composed of a single RNN layer before the output layer; these are labeled RNN1 and RNN2 and have, respectively, 60 and 100 neurons in the hidden layer. The last architecture, RNND, differs from both previous architectures in that it implements three dense layers after the RNN layer, with 100 neurons in each layer. Of the three architectures using LSTM, two start by stacking two LSTM layers. LSTM1 uses 32 neurons in both layers, while LSTMD uses 128 neurons; similar to RNND, LSTMD also implements dense layers before the output layer, with 32 neurons in the first layer and 10 neurons in the second. The BLSTM1 architecture is composed of a single BLSTM layer with 12 neurons followed by a softmax layer.

Chapter 5

Model Evaluation

This chapter first describes the test environment and preprocessing methods applied to the dataset, followed by the metrics measured during analysis and a description of how the hyperparameter optimization is implemented. The model evaluation itself is then presented, with data from both before and after hyperparameter optimization; this evaluation is divided into two sections, each focusing on one of the datasets. Finally, the throughput of the models that presented the best performance for each architecture is measured, as well as the influence of quantization on the models' throughput and accuracy.

5.1 Test environment and dataset preprocessing

All architectures in Table 4.1 are implemented using PyTorch[1], while hyperparameter tuning uses the Ray[2] tuning library. The experiments involving accuracy, precision, recall and loss, as well as the hyperparameter tuning, are performed on servers with Ubuntu 22.04 operating system, Intel i5-9600K processor with 6 cores, at least 32 GB of DDR4 RAM, and NVIDIA RTX GPU with 4,352 CUDA cores and 11 GB of memory. The throughput tests with the optimized and quantized models are performed on an NVIDIA Jetson Nano Developer Kit with Ubuntu 18.04 OS, ARM Cortex-A57 processor with 4 cores, 4 GB of LPDDR4 RAM, and NVIDIA GPU with 128 CUDA cores.

As the models' evaluation is performed using multiclass classification, BoT-IoT's *attack* and *category* features are removed during preprocessing, and *subcategory* is replaced with a *class* feature where each attack type is encoded to an integer. For the N-BaIoT dataset, a new *class* feature is created and filled based on each of N-BaIoT CSVs' filenames, also employing an integer value for each attack type. Additionally, features that might interfere with the execution of the algorithms are

[1]https://pytorch.org/
[2]https://docs.ray.io

removed during preprocessing. This includes features that record information in strings, such as the *saddr*, *daddr*, *proto*, *flgs*, and *state* features for the BoT-IoT dataset, as well as features containing redundant information, like the *HH_jit_-weight* feature of the N-BaIoT dataset, that contains the same information as *HH_-weight*. Empty features and features composed entirely of a single value are also removed; after preprocessing, the BoT-IoT dataset ended up with 35 features, while the N-BaIoT dataset remained with 115 features.

Both datasets are balanced using the SMOTE [39] algorithm so that all 11 classes have an equal amount of network traffic samples. SMOTE's "not majority" sampling strategy is used, which creates samples for all classes except the majority until all have the same number of samples; the default value of 5 k-neighbors is used during the execution of the algorithm. Next, min-max normalization of the datasets is performed so that all values are within a [0,1] range. Normalizing the data is important so that the gradient descent method, used during training, reaches convergence faster. Since the N-BaIoT dataset contains 115 features compared to BoT-IoT's 35 features, a balanced subset totaling 1,000,000 samples is extracted from the N-BaIoT dataset to reduce training time, while the balanced BoT-IoT dataset is used in its entirety. The datasets are then subdivided into a training set, consisting of 5,567,752 samples for BoT-IoT and 699,597 samples for N-BaIoT, and a test set, consisting of 2,387,934 samples for BoT-IoT and 300,403 samples for N-BaIoT.

All experiments use the Adam optimizer; the RNN and LSTM layers use the *tanh* activation function, while the MLP layers use the linear activation function. For experiments before hyperparameter optimization, all other hyperparameters use the PyTorch version 2.0.1 default values. The model training process uses K-fold cross-validation, with K equal to 5, to verify the generalizability of each model.

5.2 Metrics and hyperparameter tuning

To assess the models' performance, the accuracy, precision, and recall values for each of the 11 classes of both datasets are obtained.

The accuracy calculates the proportion of correctly classified samples compared to the total number of samples and gives a general idea of the model's performance. This metric is defined as the number of correct classifications (that is, true positives and true negatives) divided by the total amount of samples. Precision calculates the ratio of true positive samples among all samples classified as positive (that is, true positives and false positives). Recall calculates the proportion of all true positive samples among all the actual positive samples (that is, true positives and false negatives). While accuracy can be represented by a single value for each model, it is necessary to obtain the precision and recall values for each class since false positives and false negatives differ by class. Their equations are

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

where TP = True Positives; TN = True Negatives; FP = False Positives; FN = False Negatives.

The training accuracy is the accuracy obtained during model training, using the training set and considering the performance obtained during the cross-validation process. The test accuracy is the best-performing model's accuracy when used to classify the test set.

The hyperparameter optimization applied is based on the grid search method, where each possible hyperparameter combination in a predefined grid is evaluated to find the set that presents the best performance. The grid, shown in Table 5.1, considers all hyperparameter values used in the source papers, including their architectures, batch sizes, epochs, and learning rates. An adjustment to the grid search method was implemented to keep the best-performing model for each architecture, allowing comparison of each proposal's performance after the optimization process. The grid search process selects the optimal model by optimizing a target metric during training; the chosen metric is the mean F1-score, as it takes into account both the precision and recall results.

Table 5.1: Grid used for hyperparameter tuning.

Hiperparameter	Values
Architectures	[MLP1,MLP2,RNN1,RNN2,RNND, LSTM1,LSTMD,BLSTM1]
Batch Size	[32,64,100,128,1000]
Epochs	[5,10,100]
Learning Rate	[1e-3,5e-4,1e-4]

5.3 Accuracy, precision, and recall results

The accuracy results for each model, before performing the hyperparameter tuning, are shown in Figures 5.1 and 5.5. A confidence interval of 95% is used for the training accuracy, considering the results obtained during cross-validation. Since it is a deterministic result that considers all test set samples, there is no confidence interval for the test accuracy.

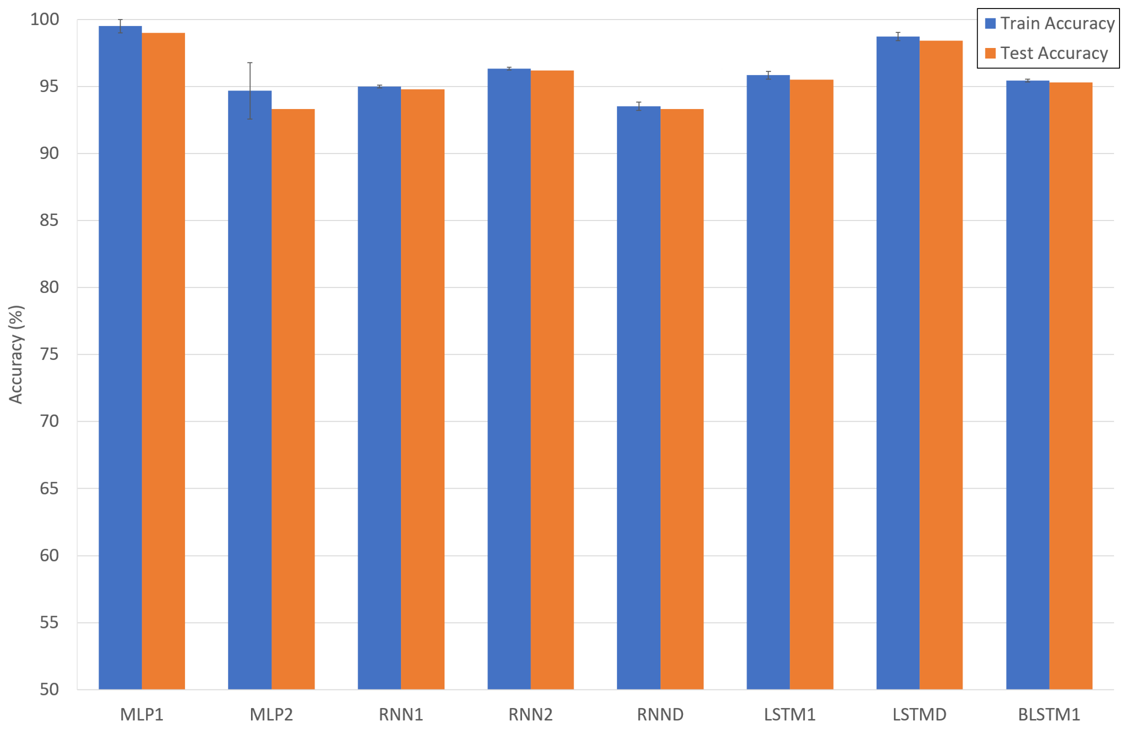


Figure 5.1: Accuracy, before hyperparameter tuning, of all evaluated neural network architectures on the BoT-IoT dataset.

5.3.1 BoT-IoT Dataset

From Figure 5.1, it is possible to observe that the non-optimized models already present satisfactory results, with all models surpassing 93% and two models exceeding 98% accuracy. Among the models, the best performance is obtained by those trained for a greater number of epochs; the only model that presents significant performance differences during training, having a large confidence interval, is MLP2. This difference potentially occurs due to the model having a large number of dense layers but being trained for only five epochs.

Figure 5.2 presents the precision and recall results obtained for each class before hyperparameter optimization. A color scale is used to represent the value of each cell, in which the color tends toward red for low values and green for high values. It can be seen that the models perform worse when detecting DoS and DDoS attacks, while they perform well in detecting legitimate traffic and other attack classes. This occurs due to the dataset balancing process; due to the very nature of these attacks, DoS and DDoS have a large number of records in the dataset. On the other hand, other attack classes are present in smaller quantities, so more attacks are algorithmically added by the SMOTE procedure. As these extra attacks are based on information present in relatively few samples, they tend to be correctly identified more often.

The difference observed between model accuracy and the precision and recall results is due to the accuracy acting as an average of the classification performance for all classes: as most classes show good accuracy, the accuracy of the model as a whole is high. On the other hand, precision and recall allow for verifying the classification performance for each class, indicating which specific classes of the model have more cases of false positives and false negatives. Through this information we can observe that, even though the MLP2 model offers between 93% and 95% accuracy, it has issues when detecting HTTP-based DoS and DDoS attacks.

		MLP1	MLP2	RNN1	RNN2	RNND	LSTM1	LSTMD	BLSTM1
Normal	Precision	99.81	98.96	99.83	99.96	98.82	99.95	99.98	99.95
	Recall	99.73	97.56	99.88	100	97.61	100	97.86	97.86
DoS-TCP	Precision	99.67	96.08	91.31	96.54	88.53	94.46	98.32	93.9
	Recall	99.97	84.61	85.87	85.3	86	84.42	98.82	84.81
DoS-UDP	Precision	100	95.23	96.14	97.5	95.53	99.42	99.96	97.09
	Recall	100	98.85	99.45	99.99	94.34	99.76	97.33	99.51
DoS-HTTP	Precision	95.49	92.44	88.71	92.51	87.31	91.41	95.23	90.93
	Recall	99.96	71.54	81.39	84.01	77.97	83.31	98.81	82.67
DDoS-TCP	Precision	100	88.99	88.02	89.04	86.27	86.09	96.6	88.23
	Recall	99.61	96.22	90.52	96.55	87.62	94.46	98.31	93.26
DDoS-UDP	Precision	100	96.66	99.4	99.99	96.6	99.77	99.57	99.51
	Recall	100	99.97	97.57	99.96	96.89	99.42	99.93	99.27
DDoS-HTTP	Precision	99.98	77.93	82.43	85.32	79.56	84.52	98.77	83.6
	Recall	95.36	94.74	90.68	93.63	89.82	92.71	95.08	92.61
Keylogging	Precision	100	100	100	100	99.98	100	99.98	100
	Recall	100	100	100	100	99.89	100	99.99	100
Data Exfiltration	Precision	99.97	99.72	99.84	100	99.89	100	99.99	100
	Recall	100	100	100	100	99.98	100	99.98	100
OS Fingerprinting	Precision	100	99.52	100	99.96	99.75	100	98.8	99.99
	Recall	100	98.63	99.48	100	99.23	100	100	99.94
Service Scan	Precision	99.73	99.75	99.88	100	97.73	100	99.08	97.9
	Recall	99.81	99.27	99.85	99.96	99.44	99.94	99.98	99.94

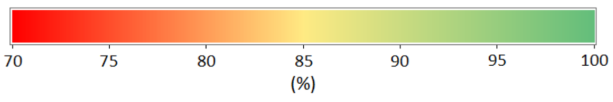
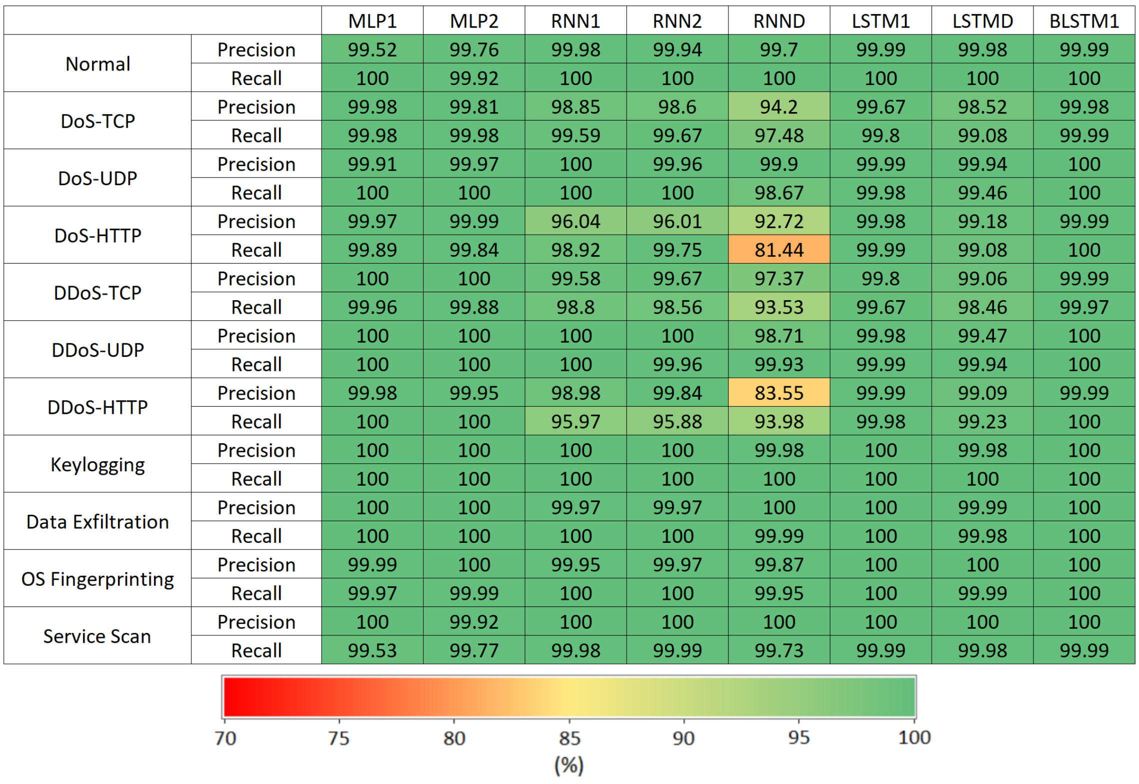


Figure 5.2: Precision and recall percentages for each evaluated class and neural network architecture before hyperparameter tuning, for the BoT-IoT dataset.

After the hyperparameter tuning, the impact of different hyperparameters on the BoT-IoT models' performance can be analyzed. The precision and recall for the optimized models are presented in Figure 5.3, while test accuracy and each model's optimal hyperparameter values are shown in Table 5.2.

Comparing Figures 5.2 and 5.3, the performance increase observed for all models is evident. As opposed to the original models, that had trouble correctly classifying DoS and DDoS attacks, most optimized models offer precision and recall values above 90% for these attacks, the only exception being the RNND model.

Comparing the results of Table 5.2 with the test accuracy of Figure 5.1, it can be seen that all architectures perform better when trained for a greater number of epochs, indicating that higher values for this hyperparameter potentially improve



		MLP1	MLP2	RNN1	RNN2	RNND	LSTM1	LSTMD	BLSTM1
Normal	Precision	99.52	99.76	99.98	99.94	99.7	99.99	99.98	99.99
	Recall	100	99.92	100	100	100	100	100	100
DoS-TCP	Precision	99.98	99.81	98.85	98.6	94.2	99.67	98.52	99.98
	Recall	99.98	99.98	99.59	99.67	97.48	99.8	99.08	99.99
DoS-UDP	Precision	99.91	99.97	100	99.96	99.9	99.99	99.94	100
	Recall	100	100	100	100	98.67	99.98	99.46	100
DoS-HTTP	Precision	99.97	99.99	96.04	96.01	92.72	99.98	99.18	99.99
	Recall	99.89	99.84	98.92	99.75	81.44	99.99	99.08	100
DDoS-TCP	Precision	100	100	99.58	99.67	97.37	99.8	99.06	99.99
	Recall	99.96	99.88	98.8	98.56	93.53	99.67	98.46	99.97
DDoS-UDP	Precision	100	100	100	100	98.71	99.98	99.47	100
	Recall	100	100	100	99.96	99.93	99.99	99.94	100
DDoS-HTTP	Precision	99.98	99.95	98.98	99.84	83.55	99.99	99.09	99.99
	Recall	100	100	95.97	95.88	93.98	99.98	99.23	100
Keylogging	Precision	100	100	100	100	99.98	100	99.98	100
	Recall	100	100	100	100	100	100	100	100
Data Exfiltration	Precision	100	100	99.97	99.97	100	100	99.99	100
	Recall	100	100	100	100	99.99	100	99.98	100
OS Fingerprinting	Precision	99.99	100	99.95	99.97	99.87	100	100	100
	Recall	99.97	99.99	100	100	99.95	100	99.99	100
Service Scan	Precision	100	99.92	100	100	100	100	100	100
	Recall	99.53	99.77	99.98	99.99	99.73	99.99	99.98	99.99

Figure 5.3: Precision and recall percentages for each evaluated class and neural network architecture after hyperparameter tuning, for the BoT-IoT dataset.

Table 5.2: Test accuracy of the optimized BoT-IoT models, and the values of the optimal hyperparameters for each model.

Architecture	Accuracy	Epochs	Batch Size	Learning Rate
MLP1	99.94%	100	100	1e-4
MLP2	99.94%	100	1000	1e-4
RNN1	99.39%	100	64	5e-4
RNN2	99.44%	100	128	5e-4
RNND	96.79%	100	128	1e-4
LSTM1	99.94%	100	32	1e-4
LSTMD	99.56%	100	1000	5e-4
BLSTM1	99.99%	100	64	5e-4

model performance for this dataset. This is valid even for architectures that originally employed a smaller amount of epochs during training. It can also be seen that the best-performing models employ a learning rate of 5e-4 or 1e-4, both lower than the default rate of 1e-3. Finally, the optimal batch size for each model varies by architecture, with architectures with fewer layers tending to show better results when using a smaller batch size. As the optimal result for most models involves using a higher amount of epochs and smaller values for the learning rate, thus taking longer to converge, it can be inferred that the BoT-IoT dataset is somewhat resistant to overfitting.

To assess how the number of epochs impacts each model's performance, the training's cross-entropy loss is also evaluated. By the loss variation of the optimized models, presented in Figure 5.4, it can be seen that most models present variable performance before approaching a loss threshold, in which the loss variation reduces significantly. The amount of epochs required to reach this threshold varies, with BLSTM1 reaching it with 40 epochs and RNN2 reaching it with more than 50 epochs. Early stopping methods can be used to conclude training the model when this threshold is reached, to save energy and processing time. The RNND model, which presented the worst performance among the optimized models, did not reach this loss threshold before the model training concluded.

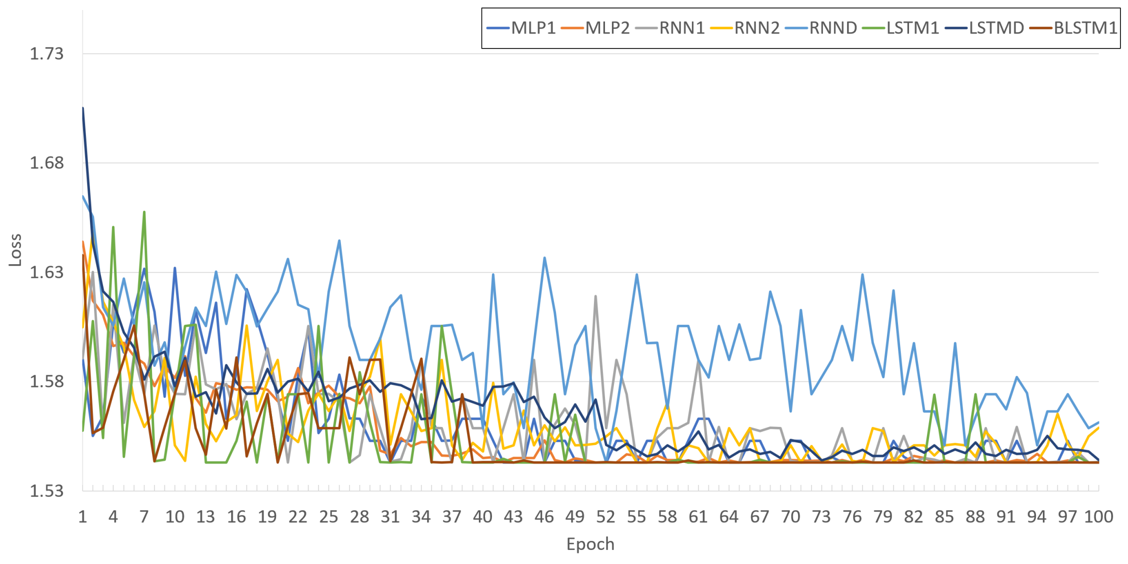


Figure 5.4: Loss per epoch of the best models obtained for each architecture after hyperparameter tuning, for the BoT-IoT dataset.

5.3.2 N-BaIoT Dataset

From Figure 5.5, it is possible to observe that the N-BaIoT models before hyperparameter optimization offer worse performance than the ones obtained with the BoT-IoT dataset. This difference may occur due to multiple factors, such as the different feature sets of each dataset, the fact that N-BaIoT models are trained with fewer samples, and the fact that N-BaIoT uses real botnet traffic instead of simulating them in virtual environments. It can be seen that models based on RNN and LSTM offer better performance, which is expected given that the N-BaIoT dataset contains temporal data in many of its features; the only exception is the RNND model, which presents significantly worse performance than the other RNN-based models.

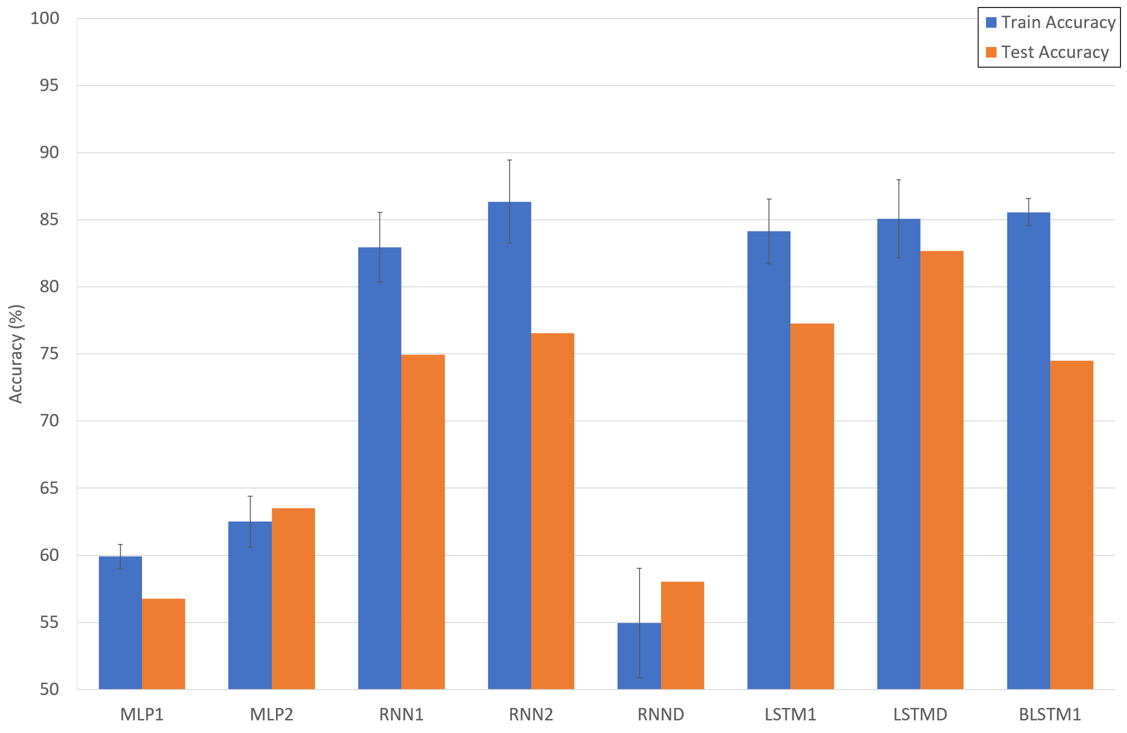
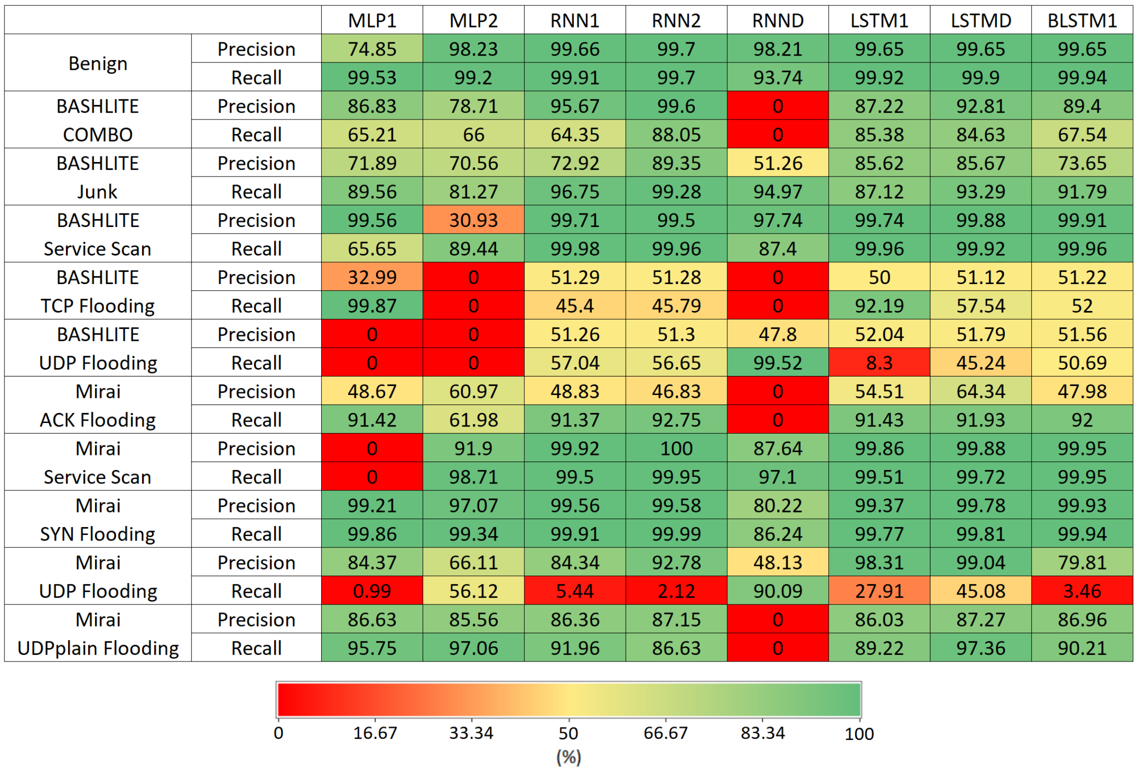


Figure 5.5: Accuracy, before hyperparameter tuning, of all evaluated neural network architectures on the N-BaIoT dataset.

Figure 5.6 presents the precision and recall results obtained for each class before hyperparameter optimization, considering all evaluated neural network architectures. A red-green color scale is used to represent the value of each cell. As seen in the figure, while most models perform satisfactorily when classifying benign traffic, they have difficulties in classifying some attack classes. In particular, BASHLITE's TCP and UDP Flooding attacks and Mirai's ACK and UDP Flooding attacks are the ones with the worst overall performance among models, indicating that they are not ideal for combating DDoS attacks. Additionally, it can be seen that the MLP1, MLP2, and RNND models obtained precision and recall equal to 0% for certain attack classes. This indicates that these models did not classify any element as belonging to those specific classes. Since these models employ multiple dense layers with a high number of neurons, this sequence of layers may potentially result in overfitting for models built using the N-BaIoT dataset. This also explains why the RNND model obtained a significantly worse performance compared to the RNN1 and RNN2 models, despite also employing RNN as its initial layer.

After the hyperparameter tuning, it is possible to observe the impact of different hyperparameters on the N-BaIoT models' performance. The precision and recall for the optimized models are presented in Figure 5.6, while test accuracy and each



		MLP1	MLP2	RNN1	RNN2	RNND	LSTM1	LSTMD	BLSTM1
Benign	Precision	74.85	98.23	99.66	99.7	98.21	99.65	99.65	99.65
	Recall	99.53	99.2	99.91	99.7	93.74	99.92	99.9	99.94
BASHLITE COMBO	Precision	86.83	78.71	95.67	99.6	0	87.22	92.81	89.4
	Recall	65.21	66	64.35	88.05	0	85.38	84.63	67.54
BASHLITE Junk	Precision	71.89	70.56	72.92	89.35	51.26	85.62	85.67	73.65
	Recall	89.56	81.27	96.75	99.28	94.97	87.12	93.29	91.79
BASHLITE Service Scan	Precision	99.56	30.93	99.71	99.5	97.74	99.74	99.88	99.91
	Recall	65.65	89.44	99.98	99.96	87.4	99.96	99.92	99.96
BASHLITE TCP Flooding	Precision	32.99	0	51.29	51.28	0	50	51.12	51.22
	Recall	99.87	0	45.4	45.79	0	92.19	57.54	52
BASHLITE UDP Flooding	Precision	0	0	51.26	51.3	47.8	52.04	51.79	51.56
	Recall	0	0	57.04	56.65	99.52	8.3	45.24	50.69
Mirai ACK Flooding	Precision	48.67	60.97	48.83	46.83	0	54.51	64.34	47.98
	Recall	91.42	61.98	91.37	92.75	0	91.43	91.93	92
Mirai Service Scan	Precision	0	91.9	99.92	100	87.64	99.86	99.88	99.95
	Recall	0	98.71	99.5	99.95	97.1	99.51	99.72	99.95
Mirai SYN Flooding	Precision	99.21	97.07	99.56	99.58	80.22	99.37	99.78	99.93
	Recall	99.86	99.34	99.91	99.99	86.24	99.77	99.81	99.94
Mirai UDP Flooding	Precision	84.37	66.11	84.34	92.78	48.13	98.31	99.04	79.81
	Recall	0.99	56.12	5.44	2.12	90.09	27.91	45.08	3.46
Mirai UDPplain Flooding	Precision	86.63	85.56	86.36	87.15	0	86.03	87.27	86.96
	Recall	95.75	97.06	91.96	86.63	0	89.22	97.36	90.21

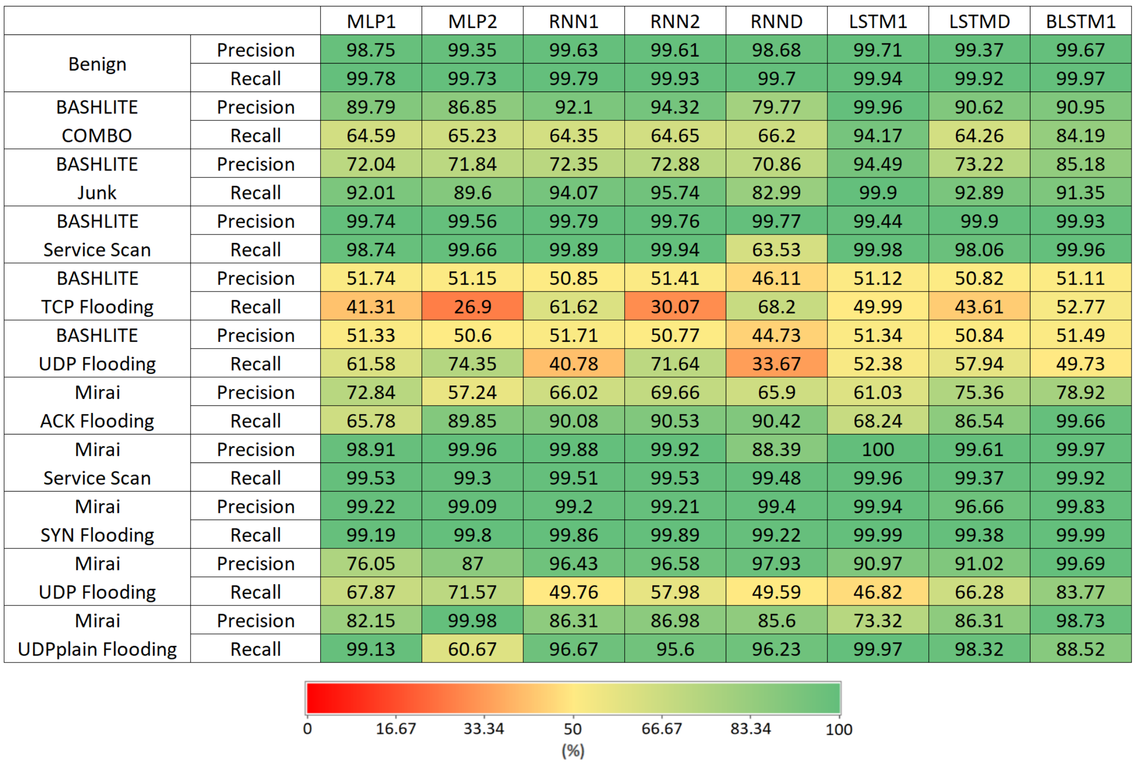
Figure 5.6: Precision and recall percentages for each evaluated class and neural network architecture before hyperparameter tuning, for the N-BaIoT dataset.

model's optimal hyperparameter values are shown in Table 5.3.

Table 5.3: Test accuracy of the optimized N-BaIoT models, and the values of the optimal hyperparameters for each model.

Architecture	Accuracy	Epochs	Batch Size	Learning Rate
MLP1	80.50%	5	64	1e-4
MLP2	79.27%	10	1000	5e-4
RNN1	80.97%	10	1000	1e-4
RNN2	81.64%	5	32	1e-3
RNND	76.79%	10	32	1e-4
LSTM1	82.44%	100	64	1e-4
LSTMD	82.66%	10	32	1e-4
BLSTM1	86.38%	10	32	1e-3

Comparing Figures 5.6 and 5.7, the performance increase observed for all models is evident. However, as opposed to BoT-IoT's results in which the performance increase led to most models achieving extremely high precision and recall values, the optimized N-BaIoT models still have trouble correctly classifying multiple attack types, particularly the BASHLITE flooding attacks. The main benefit obtained from hyperparameter optimization is that all classes were correctly identified, thus



		MLP1	MLP2	RNN1	RNN2	RNND	LSTM1	LSTMD	BLSTM1
Benign	Precision	98.75	99.35	99.63	99.61	98.68	99.71	99.37	99.67
	Recall	99.78	99.73	99.79	99.93	99.7	99.94	99.92	99.97
BASHLITE COMBO	Precision	89.79	86.85	92.1	94.32	79.77	99.96	90.62	90.95
	Recall	64.59	65.23	64.35	64.65	66.2	94.17	64.26	84.19
BASHLITE Junk	Precision	72.04	71.84	72.35	72.88	70.86	94.49	73.22	85.18
	Recall	92.01	89.6	94.07	95.74	82.99	99.9	92.89	91.35
BASHLITE Service Scan	Precision	99.74	99.56	99.79	99.76	99.77	99.44	99.9	99.93
	Recall	98.74	99.66	99.89	99.94	63.53	99.98	98.06	99.96
BASHLITE TCP Flooding	Precision	51.74	51.15	50.85	51.41	46.11	51.12	50.82	51.11
	Recall	41.31	26.9	61.62	30.07	68.2	49.99	43.61	52.77
BASHLITE UDP Flooding	Precision	51.33	50.6	51.71	50.77	44.73	51.34	50.84	51.49
	Recall	61.58	74.35	40.78	71.64	33.67	52.38	57.94	49.73
Mirai ACK Flooding	Precision	72.84	57.24	66.02	69.66	65.9	61.03	75.36	78.92
	Recall	65.78	89.85	90.08	90.53	90.42	68.24	86.54	99.66
Mirai Service Scan	Precision	98.91	99.96	99.88	99.92	88.39	100	99.61	99.97
	Recall	99.53	99.3	99.51	99.53	99.48	99.96	99.37	99.92
Mirai SYN Flooding	Precision	99.22	99.09	99.2	99.21	99.4	99.94	96.66	99.83
	Recall	99.19	99.8	99.86	99.89	99.22	99.99	99.38	99.99
Mirai UDP Flooding	Precision	76.05	87	96.43	96.58	97.93	90.97	91.02	99.69
	Recall	67.87	71.57	49.76	57.98	49.59	46.82	66.28	83.77
Mirai UDPplain Flooding	Precision	82.15	99.98	86.31	86.98	85.6	73.32	86.31	98.73
	Recall	99.13	60.67	96.67	95.6	96.23	99.97	98.32	88.52

Figure 5.7: Precision and recall percentages for each evaluated class and neural network architecture after hyperparameter tuning, for the N-BaIoT dataset.

no attack type ended up receiving 0% precision and recall as previously observed.

Comparing the results of Table 5.3 with the test accuracy of Figure 5.5, it's evident that, unlike what was observed in the models obtained with the BoT-IoT, most of the optimal N-BaIoT models were obtained using a smaller number of epochs during training. It can also be seen that the optimal batch size of the models tends to be smaller than the ones observed in the optimized BoT-IoT models. These factors indicate that the N-BaIoT dataset suffers more easily from the effects of overfitting compared to the BoT-IoT dataset, and should use smaller hyperparameter values during training. As most optimized N-BaIoT models employ a small number of epochs, the loss threshold is not reached for most models as the model training concludes too early.

5.4 Processing throughput results

All throughput experiments are performed with the optimized models, using the hyperparameter values presented in Tables 5.2 and 5.3. Table 5.4 presents the processing throughput of each model, called "Base Throughput", obtained by dividing the number of test set samples by the time each model takes to classify all samples. The experiment uses an NVIDIA Jetson Nano Developer Kit to evaluate the classification models' performance when operating on an edge device and considers a confidence level of 95%.

As observed in Table 5.4, the biggest impact on throughput is caused by the model's architecture; each model presents similar processing speeds even when applied to different datasets. Processing speed varies greatly depending on the model, varying from a minimum of 227.07 up to 1,058.65 samples classified per second. The BLSTM1 model, which presented the best classification performance for both datasets, shows an average throughput of 614.54 samples per second. Network usage of IoT devices varies by device type, with TCP streams from IoT devices sending an average of 10 to 3,000 packets per second [11]. Since network flows extracted by Open Argus commonly contain hundreds of packets per flow, a processing rate of more than 500 flows per second is sufficient to handle the standard networking requirements of IoT devices. For models with lower processing capacity and for environments with higher traffic requirements, it is necessary to evaluate the performance of each model to ensure that it fits the environment's requirements.

5.4.1 Influence of quantization

Quantization can be applied to the optimized models to reduce their size and potentially improve processing throughput [40]. Quantization is a technique that converts model weights, usually 32-bit floating points, to less accurate representations using fewer bits. This allows models to run more efficiently but may harm model classification accuracy. One of the methods of implementing quantization is called Post-Training Quantization (PTQ), where a quantized model is obtained using an existing model as a base. For this work, PTQ is applied to the models using PyTorch's quantization API so that layer weights are calculated using 8-bit integers.

The PTQ results are also displayed in Table 5.4, under the name "Quantized Throughput"; from the table, it can be seen that 8-bit quantization offers up to 3.67% throughput improvement for the quantized model. While a positive result, the performance improvement is not significant compared to each model's base performance.

Table 5.5 presents the influence of quantization on model accuracy; the accuracy difference is obtained by subtracting the post-quantization accuracy from the original accuracy. Thus, positive values indicate that there has been a performance loss in the quantized model, while negative values indicate that there has been a performance gain.

As seen from the table, the influence of 8-bit PTQ was negligible on most models' accuracy, exceeding 0.06% in only 4 of the 16 evaluated models. The overall results indicate that, while quantization does influence model performance, its im-

Table 5.4: Throughput in samples per second of the optimized models, using the hyperparameters presented in Tables 5.2 and 5.3.

Architecture	Dataset	Base Throughput (samples/s)	Quantized Throughput (samples/s)
MLP1	BoT-IoT	1,058.65 ± 19.79	1,087 ± 20.64
	N-BaIoT	1,009.43 ± 9.53	1,042.14 ± 10.51
MLP2	BoT-IoT	946.52 ± 11.22	981.05 ± 16.65
	N-BaIoT	933.72 ± 13.39	961.48 ± 15.36
RNN1	BoT-IoT	1,037.51 ± 24.32	1,068.23 ± 24.17
	N-BaIoT	1,010.14 ± 26.57	1,033.93 ± 29.75
RNN2	BoT-IoT	992.98 ± 6.39	1,027.41 ± 2.58
	N-BaIoT	960.92 ± 15.1	993.75 ± 17.96
RNND	BoT-IoT	777.74 ± 5.61	795.44 ± 5.52
	N-BaIoT	752.08 ± 9.58	779.69 ± 5.38
LSTM1	BoT-IoT	577.05 ± 6.57	587.55 ± 1.80
	N-BaIoT	517.99 ± 5.6	530.27 ± 9.96
LSTMD	BoT-IoT	227.31 ± 14.8	227.53 ± 14.74
	N-BaIoT	227.07 ± 0.17	229.9 ± 0.71
BLSTM1	BoT-IoT	614.54 ± 2.81	625.36 ± 5.1
	N-BaIoT	578.77 ± 7.91	595.75 ± 3.15

Table 5.5: Influence of quantization on the optimized models' accuracy.

Architecture	Dataset	Accuracy Difference (%)
MLP1	BoT-IoT	0.00
	N-BaIoT	0.00
MLP2	BoT-IoT	0.00
	N-BaIoT	0.00
RNN1	BoT-IoT	-0.03
	N-BaIoT	0.01
RNN2	BoT-IoT	0.03
	N-BaIoT	0.15
RNND	BoT-IoT	0.54
	N-BaIoT	0.03
LSTM1	BoT-IoT	0.06
	N-BaIoT	-0.02
LSTMD	BoT-IoT	3.05
	N-BaIoT	-0.02
BLSTM1	BoT-IoT	0.27
	N-BaIoT	-0.02

pact is overall small. Quantized models present, on average, 2.56% faster processing throughput at the cost of 0.25% lower accuracy.

Chapter 6

DL-SAFE: Deep Learning-based SAFeguard for Edge botnet detection

This chapter details DL-SAFE, an IDS for real-time detection of botnets for edge devices. Initially, the proposed architecture is described, detailing the four modules that comprise the tool and their respective functions. Afterward, we describe how a prototype was implemented using open-source tools and how its performance was evaluated with simulated DDoS attacks.

6.1 Proposed architecture

The proposed architecture consists of four modules: the model training module, the data capture module, the data processing module, and the data classification module. Figure 6.1 illustrates these modules.

The model training module is responsible for creating the classification models used by the data classification module. The training process starts by preprocessing the BoT-IoT dataset, removing incomplete, null, or redundant data. The data is then split into two sets: a training set, containing 70% of the data, and a test set containing the remaining 30%. The training set is used to train the classification models; this training uses K-fold cross-validation, with a K value of 5. The neural network architectures presented in Table 4.1 are set during training, while the hyperparameters are optimized using the grid search method; this way, it is possible to obtain the optimal hyperparameters for each evaluated architecture. After obtaining the models, their classification performance is evaluated by the offline evaluation module using the test set, acquiring performance metrics like accuracy, precision, recall, F1-Score, and loss. Unlike other modules, the model training module does not need to be active during real-time traffic classification; thus, it is possible to train the models in advance before using the tool to classify traffic data in real-time.

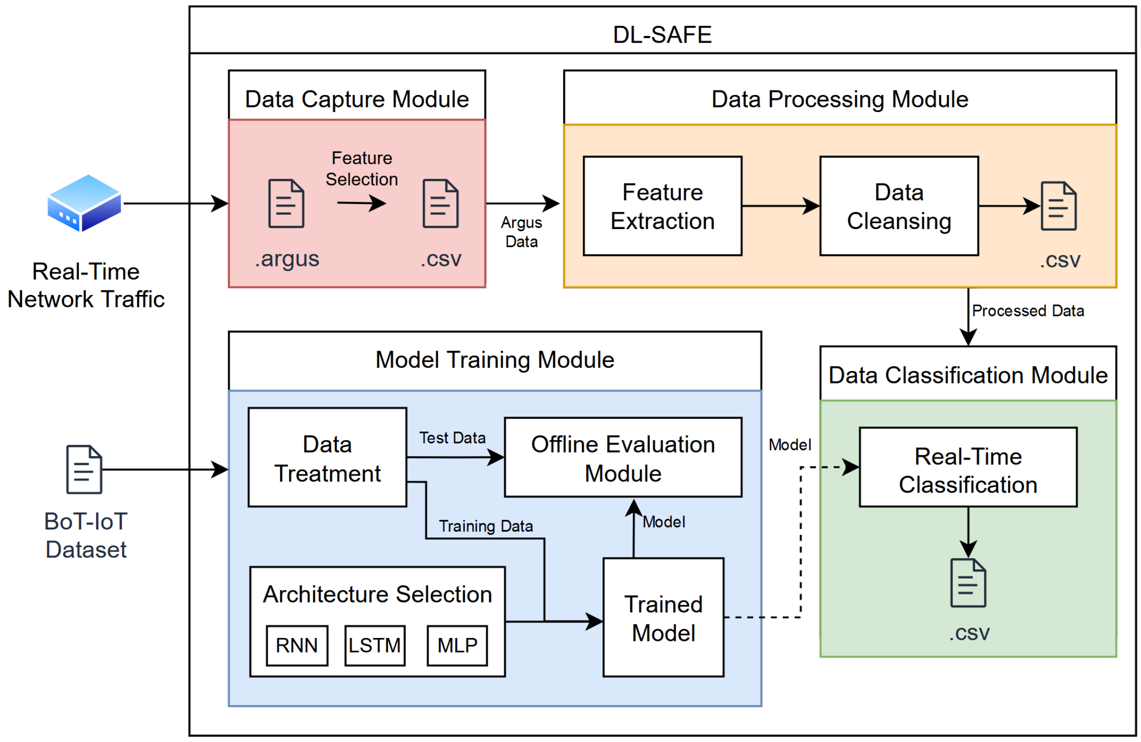


Figure 6.1: DL-SAFE's architecture, presenting the four modules.

The data capture module is implemented through Open Argus, the same tool used to create the BoT-IoT dataset. Open Argus reads network traffic and converts the data into an ARGUS file. Using Open Argus' methods, a CSV document is extracted from the ARGUS file containing the flows and their respective features in a format readable by the data processing and data classification modules. The data capture module groups network data using a configurable time window, generating a CSV file every five seconds by default.

The data processing module reads the CSV file generated by the data capture module, extracts additional features so that the final set has the same feature set used during training, and performs data cleaning using the same method implemented by the training module. This module waits until it receives a CSV file, processes this data, and sends the resulting CSV file to the data classification module.

Finally, the data classification module receives the processed data in CSV format, runs the selected classification model, and saves the classification results for further analysis by the user in a CSV file. The user can employ the classification model in its regular format or use the 8-bit post-training quantization (PTQ) format to obtain better throughput in exchange for slightly lower classification performance.

6.2 DL-SAFE Implementation

A prototype of the tool using the proposed architecture was developed using the Python language and Shell Scripts. The prototype's code, as well as the wiki explaining how to install, configure, and execute the tool, can be accessed on *https://github.com/GTA-UFRJ-team/neuralnetwork-IoT*.

A shell script coordinates the modules; this script executes relevant Python code, moves and deletes files, records the classification results, implements an IP filter, defines the time window size, and runs the necessary Open Argus commands. Each main module is implemented as follows:

- **Data Capture Module**: This module is implemented using the Open Argus system, as described in the previous section. The necessary Open Argus calls are made by the shell script that orchestrates the prototype.

- **Model Training Module**: This module is implemented as the "model_training.py" file. The training is mainly implemented using the PyTorch framework, with hyperparameter optimization being accomplished through the Ray Tune library. The Scikit-Learn library is used for cross-validation and data normalization, while the Pandas library is used to read BoT-IoT's CSV files.

- **Data Processing Module**: This module is implemented as the "csv_processing.py" file. Both the feature extraction and the data cleansing are done using the Pandas and Numpy libraries.

- **Data Classification Module**: This module is implemented as the "main_tool.py" file. Similar to the data training module, the classification is implemented using the PyTorch framework. Scikit-Learn is used for data normalization, and Pandas is used to read the treated CSV files received from the data processing module. The Watchdog library is used to monitor files, so that classification is performed as soon as the processed data is received from the data processing module. The implemented classification is based on the same code used in Chapter 5 to obtain throughput results, and thus presents a similar processing performance to that observed in Table 5.4 for the BoT-IoT models.

When the classification model detects an attack, the traffic data is added to the "detection_results.csv" file. The information recorded for each attack is the source IP, destination IP, source port, destination port, protocol, and attack type. The possible attack categories are the same 10 implemented by the BoT-IoT dataset.

6.3 Performance analysis

Two experiments are carried out to evaluate the tool's performance: a throughput test, evaluating the rate at which Open Argus can extract features from network traffic, and a classification test, in which DDoS attacks are simulated to verify the tool's ability to identify attacks in real time.

6.3.1 Open Argus throughput

The data capture module must be able to extract features from IoT network traffic in real-time. To avoid creating a bottleneck, this process must be faster than the classification itself, whose performance was presented in Table 5.4.

We use the PCAP files provided by the BoT-IoT dataset to evaluate Open Argus' ability to convert network traffic into ARGUS files during an attack. These files contain the simulated attacks used when creating the dataset. The analysis evaluates the processing capacity of Open Argus in samples per second, taking into account the time required to process each PCAP and the number of records present in the resulting file. This analysis was done separately for each attack class, to ascertain whether certain attacks significantly affect Argus' performance, and considers a confidence interval of 95%. The results are presented in Table 6.1.

Table 6.1: Throughput of Open Argus in samples per second, by evaluating BoT-IoT's PCAPs containing synthetic attacks.

Attack Class	Throughput (samples/s)
DoS-HTTP	19,279.75 ± 29.52
DoS-TCP	199,319.49 ± 4,491.73
DoS-UDP	169,343.67 ± 449.66
DDoS-HTTP	13,495.35 ± 67.67
DDoS-TCP	164,221.74 ± 1,661.27
DDoS-UDP	112,536.61 ± 380.61
Data Exfiltration	260.31 ± 0.22
Keylogging	1,201.77 ± 2.58
OS Fingerprinting	57,759 ± 386.94
Service Scan	22,345.829 ± 37.55

As seen in Table 5.4, the data classification module classifies an average of 614.54 samples per second. By analyzing the results in Table 6.1, it is evident that Open Argus's throughput significantly exceeds that value for most classes, indicating that the data capture module will not act as a bottleneck on DL-SAFE's performance. The only exception is for attack traffic belonging to the Data Exfiltration class, which achieves an average of only 260.31 samples per second.

6.3.2 DDoS detection

To perform a classification experiment in real-time, the MHDDoS[1] tool was used to simulate distributed denial of service (DDoS) attacks. Three types of attacks were performed: UDP flood, HTTP flood, and SYN flood. We evaluate how many network flows are correctly classified as denial of service attacks by DL-SAFE by comparing the tool's detection history with the Open Argus log. This log contains all network information extracted by Open Argus before any additional processing. As executing attacks using MHDDoS requires defining a target IP address and port, it is possible to obtain the ground truth by filtering the Open Argus log. Based on these results, the accuracy, precision, and recall values are calculated for each attack type. These results are obtained using the optimized BLSTM1 model, as it offers the best performance among the BoT-IoT models. The classification results are presented in Table 6.2.

Table 6.2: DL-SAFE's accuracy, precision, and recall when classifying various Distributed Denial of Service (DDoS) attacks in real-time.

Attack Type	Accuracy	Precision	Recall
UDP Flood	98.68%	99%	99.68%
HTTP Flood	99.08%	99.44%	99.63%
SYN Flood	86.65%	96.13%	89.78%

The results demonstrate that DL-SAFE offers performance greater than 98% for all evaluated metrics when detecting denial of service methods that are present in its training data. As can be seen, the tool achieves precision and recall values greater than 99% for the UDP and HTTP Flood attacks, both attacks present on the BoT-IoT dataset. On the other hand, performance is inferior when detecting DDoS attacks not present on the training data, as seen with the SYN Flood attack. While precision remains above 96%, the inferior recall indicates that a higher amount of DDoS traffic is erroneously classified as legitimate.

[1]https://github.com/MatrixTM/MHDDoS

Chapter 7

Conclusion

This work presented a performance evaluation of multiple neural network architectures applied to the detection of botnets using the BoT-IoT and N-BaIoT datasets. Eight neural network architectures were implemented using the PyTorch framework, and metrics such as accuracy, precision, and recall were obtained for each model. Next, the hyperparameters of the implemented architectures were optimized using the grid search method, using F1-Score as the target metric while also evaluating the loss variation of each model during training. Finally, the models were run on an NVIDIA Jetson Nano Developer Kit to verify performance on an edge device, which is the target environment for using the models. From the obtained results, it was possible to observe that the initial BoT-IoT models offer satisfactory classification performance, exceeding 93% accuracy in all evaluated architectures. It was also possible to achieve 99% accuracy in seven of the eight models after hyperparameter optimization, demonstrating this step's importance when building classification models. On the other hand, the N-BaIoT models' performance was noticeably inferior: before hyperparameter optimization, the best-performing model had less than 90% accuracy, and three of the evaluated models presented an accuracy inferior to 65%. Optimization improved the models' overall performance, with most models surpassing 80% accuracy; while the final performance is far from ideal, it was possible to observe the influence of different architectures and hyperparameters on a dataset that is prone to overfitting.

Additionally, this work proposes and implements DL-SAFE, an IDS for real-time traffic classification in edge environments built using Open Argus and PyTorch. In addition to classifying traffic in real-time, the tool also allows the user to build and test neural network architectures using 3 types of layers. From the obtained results, it was possible to observe that DL-SAFE offers high precision and recall when classifying known attack classes. The throughput results also demonstrate that most models are capable of handling the network traffic requirements of IoT devices, as Open Argus itself does not act as a bottleneck when extracting features in real-time.

The future steps include optimizing the execution of the models, aiming to use them more efficiently in environments with multiple IoT devices. For the deeper architectures, a possible model optimization method is to apply an early exit technique. If the predictive model indicates in a few layers that a given sample has a high probability of belonging to a certain class, the final prediction can be made in advance. Other future work includes evaluating different architectures and layer types not covered in this work, aiming to offer a more in-depth analysis of their impact on the performance of different models.

References

[1] KORONIOTIS, N., MOUSTAFA, N., SITNIKOVA, E., et al. "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset", *Future Generation Computer Systems*, v. 100, pp. 779–796, 2019.

[2] SATYAJIT SINHA. "State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally. Available at: *https://iot-analytics.com/number-connected-iot-devices/*." 2023. Last access: Nov. 15th, 2023.

[3] NESHENKO, N., BOU-HARB, E., CRICHIGNO, J., et al. "Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on Internet-scale IoT exploitations", *IEEE Communications Surveys & Tutorials*, v. 21, n. 3, pp. 2702–2733, 2019.

[4] CATALIN CIMPANU. "Microsoft said it mitigated a 2.4 Tbps DDoS attack. Available at: *https://therecord.media/microsoft-said-it-mitigated-a-2-4-tbps-ddos-attack-the-largest-ever/*." 2021. Last access: Nov. 15th, 2023.

[5] ALICIA HOPE. "Russian Internet Giant Yandex Wards off the Largest Botnet DDoS Attack in History. Available at: *https://www.cpomagazine.com/cyber-security/russian-internet-giant-yandex-wards-off-the-largest-botnet-ddos-attack-in-history/*." 2021. Last access: Nov. 15th, 2023.

[6] JONATHAN GREIG. "Google says it stopped the largest DDoS attack ever recorded in June. Available at: *https://therecord.media/google-says-it-stopped-the-largest-ddos-attack-ever-recorded-in-june/*." 2022. Last access: Nov. 15th, 2023.

[7] VORMAYR, G., ZSEBY, T., FABINI, J. "Botnet communication patterns", *IEEE Communications Surveys & Tutorials*, v. 19, n. 4, pp. 2768–2796, 2017.

[8] ATLAM, H. F., WILLS, G. B. “IoT security, privacy, safety and ethics”, *Digital twin technologies and smart cities*, pp. 123–149, 2020.

[9] MEIDAN, Y., BOHADANA, M., MATHOV, Y., et al. “N-baiot—network-based detection of iot botnet attacks using deep autoencoders”, *IEEE Pervasive Computing*, v. 17, n. 3, pp. 12–22, 2018.

[10] CYBERSECURITY VENTURES. “2022 Official Cybercrime Report. Available at: *https://s3.ca-central-1.amazonaws.com/esentire-dot-com-assets/assets/resourcefiles/2022-Official-Cybercrime-Report.pdf.*” 2022. Last access: Nov. 15th, 2023.

[11] MAINUDDIN, M., DUAN, Z., DONG, Y. “Network Traffic Characteristics of IoT Devices in Smart Homes”. In: *International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–11, 2021.

[12] KUMAR, A., SHRIDHAR, M., SWAMINATHAN, S., et al. “Machine learning-based early detection of IoT botnets using network-edge traffic”, *Computers & Security*, v. 117, pp. 102693, 2022.

[13] POKHREL, S., ABBAS, R., ARYAL, B. “IoT Security: Botnet detection in IoT using Machine learning”, *CoRR*, v. abs/2104.02231, 2021.

[14] ANDREONI LOPEZ, M., MATTOS, D. M., DUARTE, O. C. M., et al. “Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data”, *Concurrency and Computation: Practice and Experience*, v. 31, n. 20, pp. e5344, 2019.

[15] FERRAG, M. A., MAGLARAS, L., MOSCHOYIANNIS, S., et al. “Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study”, *Journal of Information Security and Applications*, v. 50, pp. 102419, 2020.

[16] CANADIAN INSTITUTE FOR CYBERSECURITY. “A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018). Available at: *https://registry.opendata.aws/cse-cic-ids2018.*” 2018. Last access: Nov. 15th, 2023.

[17] FERRAG, M. A., MAGLARAS, L. “DeepCoin: A novel deep learning and blockchain-based energy exchange framework for smart grids”, *IEEE Transactions on Engineering Management*, v. 67, n. 4, pp. 1285–1297, 2019.

[18] SHARAFALDIN, I., LASHKARI, A. H., GHORBANI, A. A. "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, v. 1, pp. 108–116, 2018.

[19] ADHIKARI, U., PAN, S., MORRIS, T., et al. "Industrial Control System (ICS) Cyber Attack Datasets. Available at: *https://sites.google.com/a/uah.edu/tommy-morris-uah/ics-data-sets*." 2019. Last access: Nov. 15th, 2023.

[20] ALKADI, O., MOUSTAFA, N., TURNBULL, B., et al. "A deep blockchain framework-enabled collaborative intrusion detection for protecting IoT and cloud networks", *IEEE Internet of Things Journal*, v. 8, n. 12, pp. 9463–9472, 2020.

[21] SARHAN, M., LAYEGHY, S., MOUSTAFA, N., et al. "NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems". In: *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 117–135, 2021. doi: 10.1007/978-3-030-72802-1_9.

[22] POPOOLA, S. I., ANDE, R., ADEBISI, B., et al. "Federated deep learning for zero-day botnet attack detection in IoT-edge devices", *IEEE Internet of Things Journal*, v. 9, n. 5, pp. 3930–3944, 2021.

[23] POPOOLA, S. I., ADEBISI, B., ANDE, R., et al. "smote-drnn: A deep learning algorithm for botnet detection in the internet-of-things networks", *Sensors*, v. 21, n. 9, pp. 2985, 2021.

[24] SAURABH, K., SOOD, S., KUMAR, P. A., et al. "LBDMIDS: LSTM Based Deep Learning Model for Intrusion Detection Systems for IoT Networks". In: *IEEE World AI IoT Congress (AIIoT)*, pp. 753–759, 2022.

[25] JAN, S., MASOODI, F., BAMHDI, A. "Effective Intrusion Detection in IoT Environment: Deep Learning Approach". In: *SCRS Conference Proceedings on Intelligent Systems*, pp. 495–502, 04 2022. doi: 10.52458/978-93-91842-08-6-47.

[26] SHAO, Z., YUAN, S., WANG, Y. "Adaptive online learning for IoT botnet detection", *Information Sciences*, v. 574, pp. 84–95, 2021.

[27] VELASCO-MATA, J., GONZÁLEZ-CASTRO, V., FIDALGO, E., et al. "Real-time botnet detection on large network bandwidths using machine learning", *Scientific Reports*, v. 13, n. 1, pp. 4282, 2023.

[28] YAN, Q., ZHENG, Y., JIANG, T., et al. "Peerclean: Unveiling peer-to-peer botnets through dynamic group behavior analysis". In: *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 316–324. IEEE, 2015.

[29] GHAFIR, I., PRENOSIL, V., HAMMOUDEH, M., et al. "Botdet: A system for real time botnet command and control traffic detection", *IEEE Access*, v. 6, pp. 38947–38958, 2018.

[30] SHARIFNYA, R., ABADI, M. "Dfbotkiller: Domain-flux botnet detection based on the history of group activities and failures in dns traffic", *Digital Investigation*, v. 12, pp. 15–26, 2015.

[31] SABA, T., REHMAN, A., SADAD, T., et al. "Anomaly-based intrusion detection system for IoT networks through deep learning model", *Computers and Electrical Engineering*, v. 99, pp. 107810, 2022.

[32] DAHOU, A., ABD ELAZIZ, M., CHELLOUG, S. A., et al. "Intrusion detection system for IoT based on deep learning and modified reptile search algorithm", *Computational Intelligence and Neuroscience*, v. 2022, 2022.

[33] KUMAR, R., KUMAR, P., TRIPATHI, R., et al. "A distributed intrusion detection system to detect DDoS attacks in blockchain-enabled IoT network", *Journal of Parallel and Distributed Computing*, v. 164, pp. 55–68, 2022.

[34] SHAFIQ, M., TIAN, Z., BASHIR, A. K., et al. "CorrAUC: A malicious bot-IoT traffic detection method in IoT network using machine-learning techniques", *IEEE Internet of Things Journal*, v. 8, n. 5, pp. 3242–3254, 2020.

[35] REY, V., SÁNCHEZ, P. M. S., CELDRÁN, A. H., et al. "Federated learning for malware detection in IoT devices", *Computer Networks*, v. 204, pp. 108693, 2022.

[36] ABU AL-HAIJA, Q., AL-DALA'IEN, M. "ELBA-IoT: an ensemble learning model for botnet attack detection in IoT networks", *Journal of Sensor and Actuator Networks*, v. 11, n. 1, pp. 18, 2022.

[37] BOCHIE, K., GILBERT, M. S., GANTERT, L., et al. "A Survey on Deep Learning for Challenged Networks: Applications and Trends", *Journal of Network and Computer Applications*, v. 194, pp. 103213, 2021. ISSN: 1084-8045.

[38] HOCHREITER, S., SCHMIDHUBER, J. "Long short-term memory", *Neural computation*, v. 9, n. 8, pp. 1735–1780, 1997.

[39] FERNÁNDEZ, A., GARCIA, S., HERRERA, F., et al. “SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary”, *Journal of artificial intelligence research*, v. 61, pp. 863–905, 2018.

[40] JACOB, B., KLIGYS, S., CHEN, B., et al. “Quantization and training of neural networks for efficient integer-arithmetic-only inference”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.