

# A Performance Evaluation of Neural Networks for Botnet Detection in the Internet of Things

Lucas C. B. Guimarães<sup>1</sup> and Rodrigo S. Couto<sup>1\*</sup>

<sup>1\*</sup>COPPE/PEE/GTA, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 21941972, RJ, Brazil.

\*Corresponding author(s). E-mail(s): [rodrigo@gta.ufrj.br](mailto:rodrigo@gta.ufrj.br);  
Contributing authors: [chagas@gta.ufrj.br](mailto:chagas@gta.ufrj.br);

## Abstract

IoT (Internet of Things) devices are fundamental to sectors such as smart homes and cities, industry 4.0, and smart grids. Despite the benefits brought by IoT, the existence of billions of devices with limited computing resources makes them ideal targets for botnets. Thus, multiple proposals have been made to detect this type of attack. However, comparing different proposals is difficult since they apply varied pre-processing methods, use different algorithms and hyperparameters, and consider distinct evaluation metrics. This paper implements and compares the performance of eight neural network architectures applied to the BoT-IoT and N-BaIoT datasets, which contain botnet attacks and labeled IoT network traffic. The models' accuracy, precision, and recall are measured, as well as the loss during model training. Afterward, the models' throughput on an edge environment is evaluated using a typical edge device, an NVIDIA Jetson Nano, while also implementing quantization and evaluating its impact on model accuracy. The results show that, after hyperparameter tuning, several BoT-IoT models exceed 99% accuracy while most N-BaIoT models surpass 80% accuracy. However, the throughput results show that the best-performing model might not scale in environments composed of a large number of IoT devices, even considering the influence of 8-bit quantization.

**Keywords:** botnet, threat detection, internet of things, deep learning

## 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 [16]. IoT has become increasingly popular, being adopted in sectors such as energy, water, transportation, health, and housing, among others. According to Cisco's Annual Internet Report, 14.7 billion IoT devices will be connected to the Internet by 2023 [9].

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 insufficient access control [20]. 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 made against Yandex and Microsoft in 2021 [2, 8]. A botnet is a network of compromised computers remotely controlled by a botmaster via a command and control server [31]. 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 the default passwords to strong ones and keeping devices up to date [4]. 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, and they 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 on 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 [19].

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 with 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 algorithm hyperparameters must also be tuned to optimize its performance.

Tuning hyperparameters is usually an expensive and time-consuming process. Initially, the architecture of the neural network 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 applied to the dataset make it difficult to compare models created in different papers.

Thus, in order to assist in the construction of future models, this paper implements and evaluates the performance of multiple neural network architectures used to detect botnet attacks in IoT networks. Two datasets are selected to build the classification models, both containing labeled IoT network traffic with botnet attacks: the BoT-IoT [16] and N-BaIoT [19] datasets. A single preprocessing method is applied to the datasets, as well as a predefined set of hyperparameters during hyperparameter tuning. 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 weights from their original 32-bit floating points to 8-bit integers. The results show that, after tuning, 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 only a single model exceeding 85% of accuracy. The throughput results indicate that, even with the improvements obtained through quantization, the network traffic of the IoT environment must be taken into account before deploying these models, as they might not meet the traffic needs of these environments.

The remainder of this paper is organized as follows. Section 2 presents the related work, while Section 3 details the datasets employed. Section 4 presents an overview of neural networks and details the chosen architectures. Section 5 describes the experiments and presents the obtained results. Finally, Section 6 concludes this article and presents future work.

## 2 Related Work

Detection of botnet attacks using machine learning is an issue that some authors have already worked on. For instance, Ferrag *et al.* conduct a survey of deep learning-based intrusion detection systems [12]. The survey considers seven deep learning models and employs them to obtain binary and multiclass classification models for the BoT-IoT and CSE-CIC-IDS2018 [7] datasets. The survey evaluates the performance of a simple architecture for each selected neural network algorithm, tuning 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 RNN model, 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 [11]. The proposed framework includes a deep learning-based scheme that uses a Recurrent Neural Network (RNN) to detect network attacks. The paper tests the proposed IDS using three datasets: BoT-IoT, CICIDS2017 [30] and a Power System dataset [1]. 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 [3]. Attacks are detected using a classification model obtained from a Bidirectional Long Short-Term Memory (BLSTM) and are evaluated using the UNSW-NB15 [27] 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 [24]. The authors evaluate multiple architectures by stacking an increasing number of fully connected layers and testing different values for hidden neurons in each layer. The architectures are tested using the BoT-IoT and N-BaIoT [19] 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 [28]. 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 [14]. 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.

Yao *et al.* propose a framework to address the challenge of noisy labels in malicious traffic detection, particularly in datasets in which mislabeled data can significantly degrade the performance of machine learning models [32]. Noisy labels refer to incorrect or imprecise labels assigned to data samples, which can occur due to human error, automated labeling inaccuracies, or ambiguities in network traffic. These mislabeled data points can mislead the learning process, causing models to learn erroneous patterns. Yao *et al.* tackle this problem by implementing a two-stage process. First, they remove the label of the noisy samples from the training data, by applying a filter. Next, they assign labels to these mislabeled samples. This assignment is performed by constructing a graph in which each node is a sample and its neighborhood is defined by a similarity metric. Hence, they assign labels to mislabeled samples based on their neighbors' labels. Their method achieves 97.27% accuracy on the BoT-IoT dataset, even with 80% label noise.

Benaddi *et al.* propose a hybrid approach to botnet detection combining a CNN with an LSTM [5]. Their model leverages CNNs to extract spatial features and LSTMs to capture temporal dependencies. Evaluated on the BoT-IoT dataset, the hybrid CNN-LSTM model achieves 99.20% accuracy and a false alarm rate of 0.80%.

Every mentioned related work builds models using the BoT-IoT dataset, which is also used in this paper. The BoT-IoT dataset is commonly chosen since it is recent and contains a large amount of labeled IoT data, including botnet attacks. This dataset, proposed in 2019, was built to simulate an IoT environment using virtual machines (VMs). A testbed composed of five IoT devices was built to represent a smart home. IoT devices are simulated using an Ubuntu operating system, while Kali Linux is used to carry out the attacks. To simulate network traffic between devices, the Ostinato<sup>1</sup> tool is used. In addition to regular traffic, the dataset also contains four categories of attacks: scanning, theft, denial of service (DoS), and distributed denial of service (DDoS). These categories are divided into 10 classes: OS Fingerprinting, Service Scan, Keylogging, Data Exfiltration, DoS-TCP, DoS-UDP, DoS-HTTP, DDoS-TCP, DDoS-UDP, and DDoS-HTTP.

Another commonly used dataset, that is also used in this paper, is the N-BaIoT dataset. Unlike BoT-IoT, which uses Kali Linux to simulate botnet attacks, N-BaIoT uses real network traffic data provided by nine IoT devices infected with two different botnets: Mirai and BASHLITE. In addition to regular traffic, the dataset contains a total of 10 attack classes, five for each botnet. The BASHLITE attacks are Service Scan, Junk (sending spam data), UDP flooding, TCP flooding, and COMBO (i.e., 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).

Regarding the N-BaIoT dataset, Javeed *et al.* propose an intelligent and interpretable intrusion detection system (IDS) for Unmanned Aerial Vehicles (UAVs) using a Hierarchical Attention-based LSTM model (H-LSTM) combined with SHAP explanations for interpretability [15]. Their approach focuses on explaining detection decisions, providing insights into which features contribute to attack detection. The model achieves high detection rates on the N-BaIoT dataset, with a detection accuracy of 99.88% and a low false positive rate.

Zhou *et al.* develop a few-shot learning mechanism for detecting malicious nodes in IoT networks using metric learning [34]. Metric learning trains models to measure similarity between data pairs, enabling classification with limited labeled samples. This approach is highly effective in scenarios with limited labeled data, achieving an accuracy of 97.67% with just five samples of malicious nodes on the N-BaIoT dataset.

Zhang *et al.* introduce a federated learning framework for detecting zero-day botnet attacks [33]. Federated learning allows multiple IoT devices to

---

<sup>1</sup><https://ostinato.org/>

collaboratively train a global model without sharing their local data, preserving privacy. In this context, one needs to weight the contribution of each device to the global model. Hence, the K-greedy aggregation algorithm proposed by Zhang *et al.* prioritizes contributions from devices that have zero-day attack records, as these devices exhibit higher detection uncertainty due to unfamiliar data, thus enhancing the global model's performance against new, unknown threats. Their model is evaluated on the N-BaIoT dataset, using a federated learning approach, and achieves 81.67% accuracy with 0.07% false alarm rate. While the accuracy is lower compared to traditional centralized detection mechanisms, this trade-off is a consequence of employing a data-privacy approach like federated learning.

While the mentioned papers regarding detection mechanisms focus on identifying botnet attacks using labeled datasets, adversarial attacks present a significant challenge to the resilience of these models. An adversarial attack manipulates input data in subtle ways to trick machine learning models, like those used for botnet detection, into making incorrect classifications. Nowroozi *et al.* propose an ensemble classifier, the 1.5-Class (cmb-classifier), which combines a 2-Class CNN with two 1-Class autoencoder classifiers to enhance resilience against adversarial attacks [22]. Their evaluation on the N-BaIoT dataset demonstrates that the cmb-classifier achieves a 0.0000% attack success rate (ASR) under the I-FGSM attack, compared to a 99% ASR for a standard CNN model.

Additionally, Nowroozi *et al.* explore the transferability of adversarial attacks, in which adversarial examples designed to deceive one detection model can also deceive other models [21]. This issue is particularly dangerous, as it allows attackers to exploit multiple models with a single crafted attack. Using the N-BaIoT dataset, they tested five types of adversarial attacks, including I-FGSM and JSMA, and found that in some cases, the attack success rate on target models reached as high as 100%. They suggest defense strategies, such as using LSTM models, which differ significantly from CNNs in structure and data processing. This difference makes it harder for adversarial examples crafted for CNNs to transfer to LSTMs, reducing the effectiveness of the attacks and improving model robustness.

The architectures in this section selected for the experiments are presented in Table 1, which is described in more detail in Section 4. This article focuses on traditional neural network architectures that are commonly used in IoT botnet detection. Approaches that incorporate more specialized techniques—such as adversarial defense mechanisms, federated learning, few-shot learning, or hybrid models such as CNN-LSTM were intentionally excluded from the experiments. These approaches, while valuable in specific contexts, introduce complexities that fall outside the scope of this paper. By focusing on widely used architectures, we provide a more controlled comparison across models, which allows for a clearer evaluation of their performance under similar conditions. This approach ensures that the results are more applicable to

environments where simplicity and performance are prioritized, without the added complexity of non-traditional methods.

**Table 1** Chosen neural network architectures and their respective hyperparameters and source papers.

Name	Architecture	Batch Size	Epochs
MLP1 [12]	Dense (100) → Dense (100) → Dense (100) → Softmax	1000	100
MLP2 [24]	Dense (100) → Dense (100) → Dense (100) → Dense (100) → Softmax	128	5
RNN1 [11]	RNN (60) → Softmax	100	5
RNN2 [12]	RNN (100) → Softmax	1000	100
RNN2 [23]	RNN (100) → Dense (100) → Dense (100) → Dense (100) → Softmax	64	10
LSTM1 [28]	LSTM (32) → LSTM (32) → Softmax	32	5
LSTM2 [14]	LSTM (128) → LSTM (128) → Dense (32) → Dense (10) → Softmax	128	100
BLSTM1 [28]	BLSTM (12) → Softmax	32	5

Unlike previous works, this paper implements and evaluates the performance of multiple neural network architectures by applying a single preprocessing method to the datasets and using the same set of hyperparameters. Evaluating the performance of varying models of the same datasets makes it possible to study the hyperparameters' influence, which can help in the selection of these hyperparameters when dealing with similar problems or datasets. Thus, the architectures are used to train classification models based on the BoT-IoT and N-BaIoT datasets and their performance is compared using the same train-test subsets. The throughput of the tuned 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.

## 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 section 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<sup>2</sup> 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<sup>3</sup> tool is used to

<sup>2</sup><https://nodered.org/>

<sup>3</sup><https://ostinato.org/>

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[17, 26, 29].

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 2, while the additional features are listed on Table 3. 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.

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.

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[24, 25]. 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 4. 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).

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

**Table 2** 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 <i>flgs</i>
proto	Textual representation of protocols present in network flow
proto_number	Numerical representation of feature <i>proto</i>
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 <i>state</i>
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

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).

**Table 3** 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

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%.

## 4 Neural Networks and Architectures

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

### 4.1 Neural Networks

Neural networks perform data transmission between artificial neurons organized in layers. Neural networks contain one input layer, one output layer, and can contain any number of hidden layers in between, with each layer containing multiple neurons [6]. There are several types of neural networks.

In the Multilayer Perceptron (MLP) algorithm each perceptron uses an activation function to connect with the others, and their output is based on the weights and inputs obtained from the previous layers. These activation functions are non-linear, allowing the implementation of non-linear models. MLP implements fully connected layers, also known as dense layers; each neuron is

**Table 4** 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

thus fully connected to the neurons of the next layer, leading to a high number of parameters and, consequently, increasing the time required for model training. The MLP is also classified as a Feedforward Neural Network (FNN), indicating that there are no cycles in the network; thus, information moves in only one direction. A Recurrent Neural Network (RNN) is a type of neural network that uses sequential or time series data. Unlike FNNs, cycles are an essential part of RNNs; the output is sent back as an additional input,

allowing the RNNs to retain information learned in previous iterations. Long Short-Term Memories (LSTMs) have additional states in the hidden layers of the neural network that control the flow of information, deciding when certain information should be forgotten; this change was made as an attempt to solve the vanishing gradient problem, a common issue present in neural networks.

## 4.2 Architectures

The architectures observed in the evaluated papers can be classified according to the type of layer used as the input layer. Considering the papers, there are three possible initial layers: dense (MLP), RNN, or LSTM. All architectures are presented in Table 1 and use softmax as the output layer for multiclass classification. This layer applies a softmax function, which converts a vector of real numbers into a probability distribution; this layer is commonly used as the last layer of neural networks since it indicates the probability that the result belongs to each possible class of the dataset. The classification considers the 10 attack classes of the BoT-IoT and N-BaIoT datasets, presented in Section 2, plus a normal traffic class, totaling 11 classes for each dataset. The number of neurons used in each layer is shown in parentheses in Table 1, which also shows the number of epochs and batch size used during training.

Among the evaluated papers, two architectures composed of multiple dense layers used in sequence are employed. The first stacks three dense layers, and the second stacks four dense layers. The architectures are labeled MLP1 and MLP2, and both use 100 neurons per layer. Three architectures starting with an RNN layer are also considered. The first two are short architectures, composed of an RNN layer followed by the softmax layer; these are called 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 all layers. Finally, both LSTM architectures stack two LSTM layers. LSTM1 uses 32 neurons in both layers, while LSTMD uses 128 neurons; similar to RNND, LSTMD also contains dense layers before the softmax 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.

## 5 Performance Analysis

This section first describes the test environment and preprocessing methods applied to the dataset, followed by the metrics considered during the analysis and how hyperparameter tuning is implemented. The performance analysis for each dataset of the evaluated architectures is then presented, followed by the results obtained after hyperparameter tuning. Finally, we evaluate the throughput of the models that presented the best performance for each considered architecture, as well as the impact of quantization on the models' throughput and accuracy.

## 5.1 Test Environment and Preprocessing

All architectures in Table 1 are implemented using PyTorch<sup>4</sup>, while hyperparameter tuning uses the Ray<sup>5</sup> tuning library; the code is available on Github<sup>6</sup>. 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 operating system, ARM Cortex-A57 processor with 4 cores, 4 GB of LPDDR4 RAM, and NVIDIA GPU with 128 CUDA cores.

During preprocessing, nonnumerical, null, or single-valued features are initially removed, as these interfere with the execution of the algorithms. BoT-IoT features are extracted using the Argus<sup>7</sup> monitoring system, which processes packets into network flows composed of a set of packets with common features. The extracted features include information such as source and destination ports, total bytes, and packets transmitted during the flow, among others. N-BaIoT features are extracted by taking behavioral snapshots over several temporal windows, summarizing traffic that has originated from the same source MAC and IP addresses towards the same destination IP. The extracted features include packet size, count, and jitter, taken from five distinct time windows.

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 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 [10] algorithm so that all 11 classes have the same amount of network traffic samples. SMOTE's "not majority" sampling strategy was selected, which creates samples for all classes except the majority until all have the same number of samples; the default value of 5 k-neighbors was also used during the execution of the algorithm. Next, the min-max normalization of the datasets is performed, resizing the

---

<sup>4</sup><https://pytorch.org/>

<sup>5</sup><https://docs.ray.io>

<sup>6</sup><https://github.com/GTA-UFRJ-team/neuralnetwork-IoT>

<sup>7</sup><https://openargus.org/>

values so that they are within the [0,1] range. Normalizing the data is important so that the gradient method, used during training, reaches convergence faster. Since the N-BaIoT dataset contains 116 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. Table 1 shows the batch size and number of epochs employed for each architecture, based on their source papers. For experiments before hyperparameter tuning, 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 assess 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 samples examined 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 number of samples. Precision calculates the ratio of true positive samples among all samples classified as positive (that is, true positives and false positives). High precision means that samples classified as positive are less likely to be negative. Recall calculates the proportion of all true positive samples among all the actual positive samples (that is, true positives and false negatives). A high recall means that positive samples are more likely to be correctly classified. 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

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

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

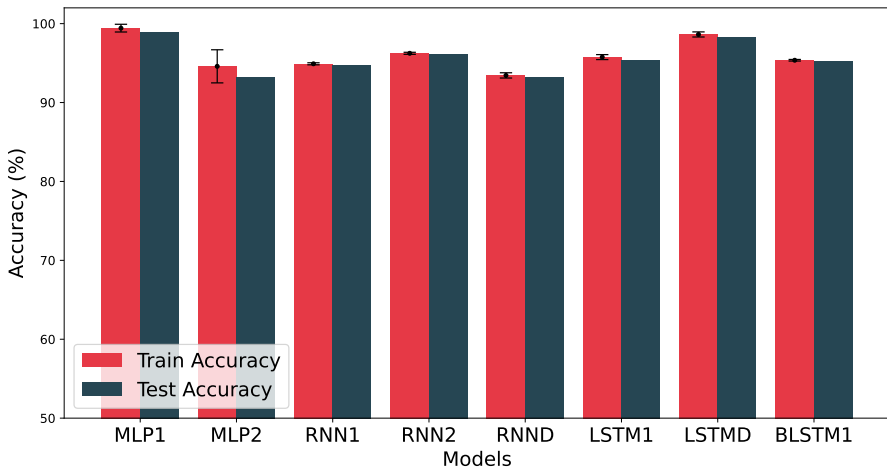
Training accuracy is the accuracy performance obtained during model training, using the training set and considering the cross-validation process.

The test accuracy is the accuracy performance of the best model obtained by the cross-validation process using the test set.

Hyperparameter tuning is implemented using the grid search method, in which each possible combination of hyperparameters in a predefined grid is evaluated to find the set that presents the best performance. The grid, shown in Table 5, considers all hyperparameter values used in the source papers, including their architectures, batch sizes, and epochs, as well as varying learning rates. Grid search is done by optimizing a target metric during hyperparameter tuning; the chosen metric is the mean F1-score, as it is the harmonic mean between precision and recall.

**Table 5** Grid used for hyperparameter tuning.

Hyperparameter	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]



**Fig. 1** Accuracy, before hyperparameter tuning, of all evaluated neural network architectures on the BoT-IoT dataset.

### 5.3 Accuracy, Precision, and Recall Results

The accuracy results for each model, before performing the hyperparameter tuning, are shown in Figures 1 and 4. Each architecture uses a certain number of epochs and batch size as stated in their source paper, according to

Table 1. For training accuracy, a confidence interval of 95% was used, considering the results obtained during cross-validation. Since it is a deterministic result, which considers all test set samples, there is no confidence interval for the test accuracy. The results of Figures 1 and 4 are discussed as follows.

### 5.3.1 BoT-IoT Dataset

From Figure 1, it is possible to observe that the models present satisfactory results, with all models reaching accuracy above 93% and two models exceeding 98%. Among the models, the best performance was obtained by those trained for a greater number of epochs, indicating that optimizing this hyperparameter can improve the performance of other models for this dataset. 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 2 presents the precision and recall results obtained for each class before hyperparameter tuning, considering all evaluated neural network architectures. A color scale was used to represent the value of each cell, in which the color tends toward red for low values and green for high values. It is possible to observe that the models perform worse in detecting DoS and DDoS attacks, while they perform well in detecting legitimate traffic and other classes of attacks contained in the dataset. 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.

After hyperparameter tuning, the impact of different hyperparameters on the BoT-IoT models' performance can be analyzed. The test accuracy of the optimized models and the respective optimal hyperparameter values are shown in Table 6.

**Table 6** Test Accuracy of the tuned 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

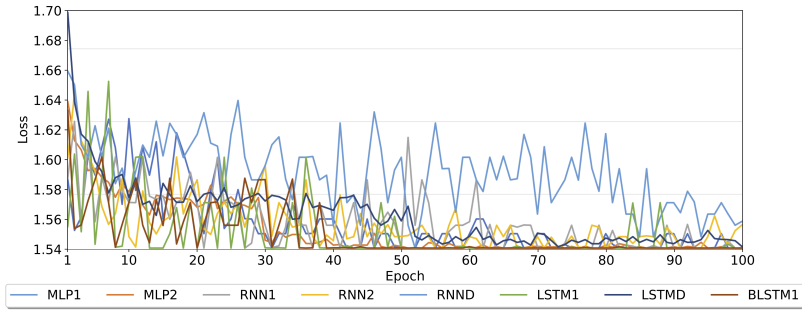
		MLP1	MLP2	RNN1	RNN2	RNND	LSTM1	LSTMD	BLSTM1
Normal	Precision	99.81	98.96	99.83	99.96	98.82	99.95	99.98	99.95
	Recall	99.73	97.56	99.88	100	97.61	100	97.86	97.86
DoS-TCP	Precision	99.67	96.08	91.31	96.54	88.53	94.46	98.32	93.9
	Recall	99.97	84.61	85.87	85.3	86	84.42	98.82	84.81
DoS-UDP	Precision	100	95.23	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

**Fig. 2** Precision and Recall percentages for each evaluated class and neural network architecture before hyperparameter tuning, for the BoT-IoT dataset.

When comparing the results of Table 6 with the test accuracy of Figure 1, it can be seen that all architectures perform better when trained for a greater number of epochs. This is valid even for those that originally used a smaller amount of epochs during training. It is also observed 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 using a smaller batch size.

To assess how the number of epochs impacts the performance of each model, the loss given by cross-entropy during training was recorded. Cross-entropy is the loss function used when fitting model weights; this function calculates the loss by comparing how far the predictions are from the correct class. The goal during training is to minimize this value; therefore, a smaller loss results in a better model.

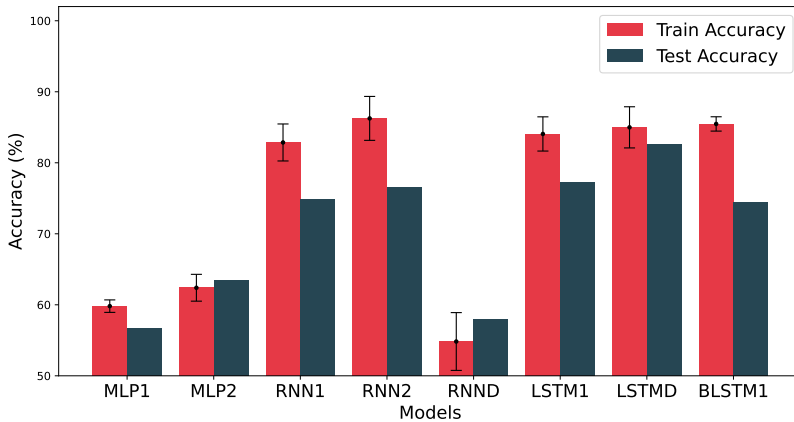
By the loss variation of the best models, presented in Figure 3, it can be seen that most models present variable performance before approaching a loss threshold, in which the loss variation reduces significantly. The amount of epochs required to reach this threshold varies, with BLSTM1 reaching it with 40 epochs and RNN2 reaching it with more than 50 epochs. Early stopping methods can be used to finish 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 for the range of epochs considered.



**Fig. 3** 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

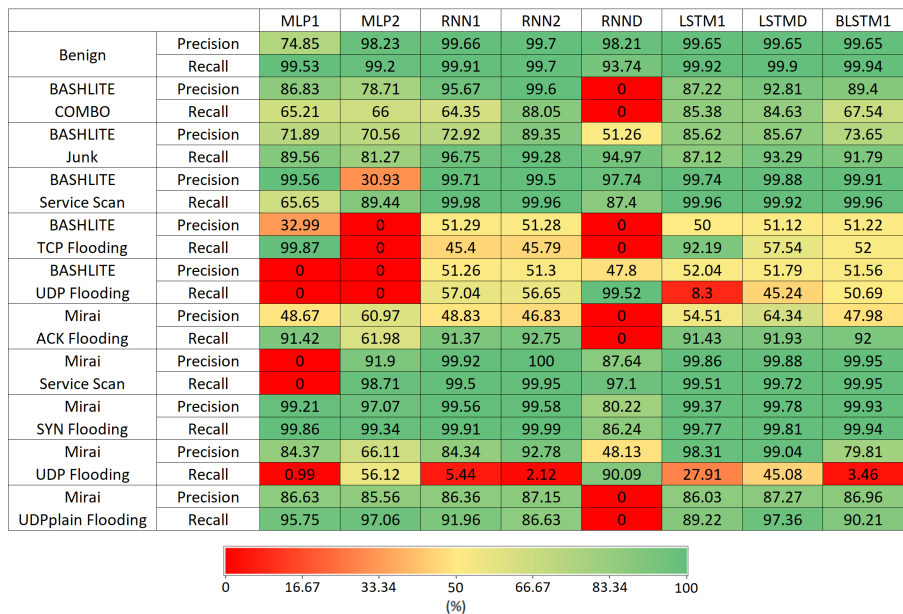
From Figure 4, it is possible to observe that the N-BaIoT models before hyperparameter tuning offer worse performance than the ones obtained with the BoT-IoT dataset. This difference is due to multiple factors, such as the different set of features used by each dataset, the fact that N-BaIoT models are trained with fewer samples, and the fact that N-BaIoT executes attacks with real botnets instead of executing them with Kali Linux 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 its features; the only exception is the RNN model, which presents significantly worse performance than the other RNN-based models.



**Fig. 4** Accuracy, before hyperparameter tuning, of all evaluated neural network architectures on the N-BaIoT dataset.

Figure 5 presents the precision and recall results obtained for each class before hyperparameter tuning, considering all evaluated neural network architectures. A similar red-green color scale as that used in Figure 2 is used to

represent the value of each cell. As seen in the figure, while most models perform satisfactorily when classifying benign traffic, the models 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 the models, indicating that they are not ideal for combating these DDoS attacks. It can be seen that some models obtained precision and recall equal to 0 for certain attacks, indicating that the model did not classify any element as belonging to that specific class. This is observed in the MLP1, MLP2, and RNN2 models; since these models employ multiple dense layers with a high number of neurons, it can be concluded that this sequence of layers potentially worsens the performance of models built using the N-BaIoT dataset. This also explains why the RNN2 model obtained significantly worse performance compared to the RNN1 and RNN2 models.



**Fig. 5** Precision and Recall percentages for each evaluated class and neural network architecture before hyperparameter tuning, for the N-BaIoT dataset.

After hyperparameter tuning, it is possible to observe the impact of different hyperparameters on the N-BaIoT models’ performance. The test accuracy of the tuned models and the respective optimal hyperparameter values are shown in Table 7.

Comparing the results of Table 7 with the test accuracy of Figure 4, it is evident that, unlike what was observed in the models obtained with 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 tuned BoT-IoT

**Table 7** Test Accuracy of the tuned 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

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 the tuned N-BaIoT models mostly employ a small number of epochs, the value of that hyperparameter is not sufficient to reach the loss threshold observed during training for the BoT-IoT models.

## 5.4 Processing Throughput Results

All throughput experiments are performed with the tuned models, using the hyperparameter values presented in Tables 6 and 7. Table 8 presents the processing throughput of each model, called “Base Throughput”, obtained by dividing the number of test set samples by the time it takes for each model to classify all samples. The experiment uses an NVIDIA Jetson Nano Developer Kit to evaluate the performance of classification models when operating on an edge device and considers a confidence level of 95%.

As observed in Table 8, 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, ranging 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 3000 packets per second [18]. Thus, although the tuned models’ throughput is sufficient to handle a limited number of IoT devices, the models might not scale in environments that have dozens or hundreds of devices, or that include devices with higher network usage.

## 5.5 Impact of Quantization

Quantization can be applied to the tuned models to reduce their size and potentially improve their processing throughput [13]. Quantization is a technique that converts model weights, usually 32-bit floating points, to less accurate

**Table 8** Throughput in samples per second of tuned models, using the hyperparameters presented in Tables 6 and 7.

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

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. PTQ is applied to the models using PyTorch’s quantization API so that layer weights are calculated using 8-bit integers instead.

The PTQ results are also shown in Table 8, under the name “Quantized Throughput”; from the table, it can be seen that the 8-bit quantization offers up to 3,67% throughput improvement depending on the quantized model. While a positive result, the performance improvement is still not enough to support IoT devices with high network consumption. Table 9 presents the impact of quantization on model accuracy; the difference in accuracy 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 impact of 8-bit PTQ was small on most models’ accuracy, exceeding 0.06% in 4 of the 16 evaluated models.

## 6 Conclusion

This paper presented a performance evaluation of multiple neural network architectures applied to botnet detection 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, a hyperparameter tuning of the implemented

**Table 9** Influence of quantization on the tuned 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

architectures was performed using the grid search method, having the mean F1-score as a target metric and also verifying the variation of the loss of each model during training. Finally, the models as well as their quantized versions were executed on an NVIDIA Jetson Nano Developer Kit to verify performance on an edge device, which is the target environment for these models.

From the obtained results it was possible to observe that, before tuning, the BoT-IoT models offer satisfactory classification performance, surpassing the test accuracy 93% in all of the eight models evaluated. On the other hand, the N-BaIoT models offered worse performance, with only a single architecture attaining accuracy superior to 80%. With hyperparameter tuning it was possible to reach 99% accuracy in seven of the eight BoT-IoT models, and accuracy superior to 80% in six of the eight N-BaIoT models, demonstrating the importance of this step during model training. The throughput results show that the models' performance when running on an edge device might not be enough to handle a higher volume of IoT network traffic, indicating that there are challenges for the scalable application of these models in IoT environments.

The next steps involve studying other aspects in which quantization can improve the models' processing capacity. For instance, quantization reduces model size while having a low impact on accuracy; thus, given the lower memory consumption per model, a possible strategy would be to run multiple models in parallel on an edge device. Another possibility to improve throughput performance would be the implementation of 4-bit and 3-bit quantization. Finally, another prospect would be to verify how different quantization methods, such as Quantization Aware Training (QAT), would impact model performance.

## 7 Author Declarations

### 7.1 Availability of data and materials

All the data utilized in this study originate from third-party datasets that are openly accessible online.

### 7.2 Competing interests

The authors declare no competing interests.

### 7.3 Funding

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, FAPERJ grants 26/010.002174/2019 and SEI-260003/004771/2021, FAPESP grants 23/00673-7 and 23/00811-0, and CNPq grant 408255/2023-4.

### 7.4 Authors' contributions

Lucas C. B. Guimarães has played a role in carrying out the experiments, drafting the initial manuscript, and analyzing the findings. Rodrigo S. Couto has contributed through reviewing the manuscript and establishing the methodology. Both authors have reviewed and consented to the final version of the manuscript.

## References

- [1] Adhikari, U., Pan, S., Morris, T., Borges, R., Beaver, J.: Industrial Control System (ICS) Cyber Attack Datasets. available at: <https://sites.google.com/a/uah.edu/tommy-morris-uah/ics-data-sets>. (2019)
- [2] 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)
- [3] Alkadi, O., Moustafa, N., Turnbull, B., Choo, K.K.R.: A deep blockchain framework-enabled collaborative intrusion detection for protecting iot and cloud networks. *IEEE Internet of Things Journal* **8**(12), 9463–9472 (2020)
- [4] Atlam, H.F., Wills, G.B.: Iot security, privacy, safety and ethics. *Digital twin technologies and smart cities* pp. 123–149 (2020)
- [5] Benaddi, H., Jouhari, M., Ibrahimi, K., Benslimane, A., Amhoud, E.: Improvement of anomaly detection system in iot networks using cnn-lstm

- approach. In: IEEE Global Communications Conference (GLOBECOM) (2023)
- [6] Bochie, K., Gilbert, M.S., Gantert, L., Barbosa, M.S.M., Medeiros, D.S.V., Campista, M.E.M.: A survey on deep learning for challenged networks: Applications and trends. *Journal of Network and Computer Applications* **194**, 103213 (2021)
- [7] Canadian Institute for Cybersecurity: A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018). available at: <https://registry.opendata.aws/cse-cic-ids2018>. (2018)
- [8] 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)
- [9] Cisco: Cisco Annual Internet Report (2018–2023). available at: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>. (2018)
- [10] Fernández, A., Garcia, S., Herrera, F., Chawla, N.V.: Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research* **61**, 863–905 (2018)
- [11] Ferrag, M.A., Maglaras, L.: Deepcoin: A novel deep learning and blockchain-based energy exchange framework for smart grids. *IEEE Transactions on Engineering Management* **67**(4), 1285–1297 (2019)
- [12] Ferrag, M.A., Maglaras, L., Moschoyiannis, S., Janicke, H.: Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications* **50**, 102419 (2020)
- [13] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: 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)
- [14] 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 (2022). DOI 10.52458/978-93-91842-08-6-47
- [15] Javeed, D., Gao, T., Kumar, P., Shoukat, S., Ahmad, I., Kumar, R.: An intelligent and interpretable intrusion detection system for unmanned

- aerial vehicles. In: IEEE International Conference on Communications (ICC) (2024)
- [16] Koroniotis, N., Moustafa, N., Sitnikova, E., Turnbull, B.: Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems* **100**, 779–796 (2019)
- [17] Kumar, R., Kumar, P., Tripathi, R., Gupta, G.P., Garg, S., Hassan, M.M.: A distributed intrusion detection system to detect DDoS attacks in blockchain-enabled iot network. *Journal of Parallel and Distributed Computing* **164**, 55–68 (2022)
- [18] 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)
- [19] Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., Elovici, Y.: N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing* **17**(3), 12–22 (2018)
- [20] Neshenko, N., Bou-Harb, E., Crichigno, J., Kaddoum, G., Ghani, N.: Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations. *IEEE Communications Surveys & Tutorials* **21**(3), 2702–2733 (2019)
- [21] Nowroozi, E., Mekdad, Y., Hajian Berenjestanaki, M., Conti, M., El Fergougui, A.: Demystifying the transferability of adversarial attacks in computer networks. *IEEE Transactions on Network and Service Management* **19**(3), 3387–3400 (2022)
- [22] Nowroozi, E., Mohammadi, M., Savas, E., Mekdad, Y., Conti, M.: Employing deep ensemble learning for improving the security of computer networks against adversarial attacks. *IEEE Transactions on Network and Service Management* **20**(2), 2096–2105 (2023)
- [23] Popoola, S.I., Adebisi, B., Ande, R., Hammoudeh, M., Anoh, K., Atayero, A.A.: smote-drrn: A deep learning algorithm for botnet detection in the internet-of-things networks. *Sensors* **21**(9), 2985 (2021)
- [24] Popoola, S.I., Ande, R., Adebisi, B., Gui, G., Hammoudeh, M., Jogunola, O.: Federated deep learning for zero-day botnet attack detection in iot-edge devices. *IEEE Internet of Things Journal* **9**(5), 3930–3944 (2021)
- [25] Rey, V., Sánchez, P.M.S., Celdrán, A.H., Bovet, G.: Federated learning for malware detection in iot devices. *Computer Networks* **204**, 108693 (2022)

- [26] Saba, T., Rehman, A., Sadad, T., Kolivand, H., Bahaj, S.A.: Anomaly-based intrusion detection system for iot networks through deep learning model. *Computers and Electrical Engineering* **99**, 107810 (2022)
- [27] Sarhan, M., Layeghy, S., Moustafa, N., Portmann, M.: 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. URL [https://doi.org/10.1007%2F978-3-030-72802-1\\_9](https://doi.org/10.1007%2F978-3-030-72802-1_9)
- [28] Saurabh, K., Sood, S., Kumar, P.A., Singh, U., Vyas, R., Vyas, O., Khondoker, R.: LBDMIDS: LSTM based deep learning model for intrusion detection systems for iot networks. In: *IEEE World AI IoT Congress (AIIoT)*, pp. 753–759 (2022)
- [29] Shafiq, M., Tian, Z., Bashir, A.K., Du, X., Guizani, M.: CorrAUC: A malicious bot-iot traffic detection method in iot network using machine-learning techniques. *IEEE Internet of Things Journal* **8**(5), 3242–3254 (2020)
- [30] Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **1**, 108–116 (2018)
- [31] Vormayr, G., Zseby, T., Fabini, J.: Botnet communication patterns. *IEEE Communications Surveys & Tutorials* **19**(4), 2768–2796 (2017)
- [32] Yao, L., Niu, W., Yuan, Q., Li, B., Zhang, Y., Zhang, X.: A robust malicious traffic detection framework with low-quality labeled data. In: *IEEE International Conference on Communications (ICC)* (2024)
- [33] Zhang, J., Liang, S., Ye, F., Hu, R.Q., Qian, Y.: Towards detection of zero-day botnet attack in iot networks using federated learning. In: *IEEE International Conference on Communications (ICC)* (2023)
- [34] Zhou, K., Lin, X., Wu, J., Bashir, A.K., Li, J., Imran, M.: Metric learning-based few-shot malicious node detection for iot backhaul/fronthaul networks. In: *IEEE Global Communications Conference (GLOBECOM)* (2022)