Computers & Security 133 (2023) 103388

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Towards a robust, effective and resource efficient machine learning technique for IoT security monitoring

Idris Zakariyya [a,*], Harsha Kalutarage [b], M. Omar Al-Kadri [c]

[a] School of Computing Science, University of Glasgow, UK
[b] School of Computing, Robert Gordon University, UK
[c] School of Computing and Digital Technology, Birmingham City University, UK

ARTICLE INFO

Keywords:
Internet of things
Deep neural networks
Cybersecurity
Resource constrained
Attack detection
Federated learning

ABSTRACT

The application of Deep Neural Networks (DNNs) for monitoring cyberattacks in Internet of Things (IoT) systems has gained significant attention in recent years. However, achieving optimal detection performance through DNN training has posed challenges due to computational intensity and vulnerability to adversarial samples. To address these issues, this paper introduces an optimization method that combines regularization and simulated micro-batching. This approach enables the training of DNNs in a robust, efficient, and resource-friendly manner for IoT security monitoring. Experimental results demonstrate that the proposed DNN model, including its performance in Federated Learning (FL) settings, exhibits improved attack detection and resistance to adversarial perturbations compared to benchmark baseline models and conventional Machine Learning (ML) methods typically employed in IoT security monitoring. Notably, the proposed method achieves significant reductions of 79.54% and 21.91% in memory and time usage, respectively, when compared to the benchmark baseline in simulated virtual worker environments. Moreover, in realistic testbed scenarios, the proposed method reduces memory footprint by 6.05% and execution time by 15.84%, while maintaining accuracy levels that are superior or comparable to state-of-the-art methods. These findings validate the feasibility and effectiveness of the proposed optimization method for enhancing the efficiency and robustness of DNN-based IoT security monitoring.

1. Introduction

The Internet of Things (IoT) has witnessed significant growth, connecting physical devices through diverse protocols to perform specific tasks. These devices utilize embedded systems such as processors, sensors, and communication hardware to collect and exchange data. Projections indicate that the global data collected by IoT devices will reach 73.1 zettabytes by 2025 (Bojan, 2022). Advancements in affordable computer chips and wireless networks have enabled the realization of IoT technology, fostering unprecedented connectivity among devices. This technology has facilitated the development of smart homes, smart cities, and various intelligent automation systems. It is estimated that approximately 125 billion devices will be interconnected by 2030 (Jenalea, 2017).

However, the proliferation of IoT devices has exposed them to cyberattacks, as attackers exploit vulnerabilities to execute various attacks when devices connect to the external world. The Mirai botnet, a well-known example, demonstrates the consequences of such attacks (Antonakakis et al., 2017). To counter these threats, integrating Artificial Intelligence (AI) with IoT systems has emerged as a solution. By leveraging AI, whether in a centralized or decentralized approach, malicious activities can be detected and thwarted effectively. However, resource-constrained IoT devices typically have limited hardware capabilities, including 32 KB to 128 KB units of Random Access Memory (RAM) and a 256 KB to 512 KB embedded flash memory footprint (Zandberg et al., 2019). These constraints pose challenges for deploying resource-intensive AI models. Consequently, it is crucial to address security challenges in IoT networks through effective and efficient detection techniques.

Recent research has demonstrated the potential of AI-based technologies, specifically Machine Learning (ML) and Deep Neural Network (DNN) approaches, for cybersecurity monitoring (Merenda et al., 2020; Vinayakumar et al., 2018). DNN-based methods have garnered particular interest due to their ability to detect attacks on various targets, including IoT devices, endpoint devices, and the cloud (Kshetri, 2021). However, a significant limitation of DNN-based approaches is

* Corresponding author.
E-mail addresses: Idris.Zakariyya@glasgow.ac.uk (I. Zakariyya), h.kalutarage@rgu.ac.uk (H. Kalutarage), omar.alkadri@bcu.ac.uk (M.O. Al-Kadri).

https://doi.org/10.1016/j.cose.2023.103388
Received 29 September 2022; Received in revised form 3 June 2023; Accepted 16 July 2023
Available online 20 July 2023

their substantial computational resource requirements for constructing models capable of providing an improved threat detection system with a multi-dimensional network security feature set (Aggarwal et al., 2018). Consequently, deploying such resource-intensive models in environments with limited computing resources, such as IoT, becomes challenging. This issue is further exacerbated in Federated Learning (FL) contexts, where on-device model learning facilitates collaborative learning among edge devices without exposing their data to the cloud or fog (Yang et al., 2019). Additionally, DNN-based detection methods can be vulnerable to adversarial samples, which pose a significant security threat.

To address these limitations and enhance the efficiency and robustness of DNN-based IoT security monitoring, this paper investigates the challenges associated with employing DNN methods in this context. The main aim of this research is to create a Resource-Efficient Deep Neural Network (REDNN) model that can effectively detect attacks on IoT networks. The objective is to achieve a comparable or improved level of accuracy compared to existing models while maintaining a desired level of resource efficiency. The study also focuses on evaluating the resilience of the proposed REDNN method against adversarial attacks. Additionally, the research aims to showcase the efficiency and accuracy of REDNN in real-time attack detection within a decentralized federated scenario. Specifically, we address the following research questions:

RQ1: How can an existing DNN be trained to create a REDNN model capable of detecting attacks on IoT networks with comparable or improved accuracy compared to baseline models, while also achieving a desired level of resource efficiency? (sections 3.2 and 5.1).
RQ2: Is the proposed REDNN method robust against adversarial attacks compared to baseline and other conventional ML models? (sections 5.2 and 5.2.3).
RQ3: Does the resulting REDNN exhibit both efficiency and accuracy in real-time detection of attacks on IoT networks within a decentralized federated scenario, showcasing better or state-of-the-art performance? (section 5.3).

To address these research questions, this study introduces the REDNN methodology, which optimizes DNN models for resource efficiency while maintaining or improving accuracy performance. Additionally, we explore the resilience of REDNN against adversarial attacks and evaluate its performance in a decentralized federated scenario. We conduct experiments using a Fully Connected Neural Network (FCNN) and eleven benchmark datasets specific to IoT environments. The experimental results demonstrate the effectiveness of REDNN in terms of attack detection and resource efficiency. Consequently, this paper presents the following contributions:

1. Introducing the REDNN methodology as a solution to the challenges associated with deploying DNN technologies efficiently in IoT environments.
2. Evaluating the resilience of REDNN against adversarial attacks, offering robust security measures for IoT devices.
3. Introducing the Resource Efficient Federated Deep Neural Network (REFDNN) methodology for training DNN models in a federated IoT security monitoring environment, providing accurate security monitoring while ensuring data privacy and reducing memory footprint and execution time.

To the best of our knowledge, this is the first attempt to examine the capabilities of DNN models for resource efficiency, robust detection, and on-device learning in the context of IoT security, utilizing a large number of benchmark datasets generated by hostile attacks on commercial IoT devices.

Throughout the paper, the term “resource-efficient” is employed to describe models that exhibit reduced memory consumption and require less time in comparison to their baseline benchmarks. Hence the term is defined in relative terms rather than absolute measurements.

Table 1
List of acronyms.

Acronym	Meaning
Acc/acc	Accuracy
AI	Artificial Intelligence
AIoT	Artificial Intelligence of Things
BFDNN	Baseline Federated Deep Neural Network
B	Byte
CPSs	Cyber-physical Systems
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DoS	Denial-of-Service
DDoS	Distributed Denial-of-Service
FCNN	Fully Connected Neural Network
FedAvg	Federated Averaging
FGSM	Fast Gradient Sign Method
FL	Federated Learning
GB	Gradient Boosting
IoT	Internet of Things
LSTM	Long Short Term Memory
MB	Megabyte
mins	Minutes
ML	Machine Learning
ms	Milliseconds
NS2	Network Simulator Version 2
OS	Operating System
PGD	Projected Gradient Descent
RAM	Random Access Memory
REDNN	Resource Efficient Deep Neural Network
REFDNN	Resource Efficient Federated Deep Neural Network
SVM	Support Vector Machine
TFLite	TensorFlow Lite
WS	Websocket

The remaining sections of this paper are structured as follows. Section 2 provides a thorough review of the related literature. Section 3 outlines the proposed methodology and describes the FL technique utilized, while Section 4 elaborates on the evaluation process. The findings and analysis are presented in Section 5. Finally, Section 6 provides concluding remarks and identifies avenues for future research. The acronyms used throughout the paper are listed in Table 1.

2. Related work

The following section presents a comprehensive review of relevant studies that delve into the exploration of AI-based detection algorithms employed for IoT security monitoring with the aim of resolving security and privacy issues that arise during the deployment of AI-based solutions in environments with limited resources. Additionally, it investigates contemporary FL approaches utilized in IoT environments with the primary objective of addressing data privacy, security, and resource efficiency concerns in realistic decentralized IoT network environments.

2.1. AI techniques for IoT security monitoring

The literature provides ample evidence of the widespread use of AI techniques to address security challenges within the IoT (Sánchez et al., 2021). Elrawy et al. (2018) recommended the development of ML and DNN-based intrusion detection systems for IoT, which have shown significant potential in the field of IoT security monitoring research. Hsu et al. (2019) proposed a framework that utilizes Support Vector Machine (SVM) to monitor IoT network security by detecting anomalous behavior. The SVM approach achieved a detection accuracy of 92.30% using simulated IoT smart homes data. Similarly, (Lopez-Martin et al., 2020) proposed an IoT network traffic forecasting technique using the stochastic Gradient Boosting (GB) classifier, which exhibited superior performance in detecting active connections compared to inactive traffic flow. In addition, (Tang et al., 2020) enhanced the Adaboost algo-

rithm to detect low-rate Distributed Denial-of-Service (DDoS) attacks in an IoT environment, achieving a detection rate of 97.06% using Network Simulator Version 2 (NS2) for model performance assessment. Furthermore, (Zhang et al., 2019) developed a DNN-based framework to detect cyber-attacks in various Cyber-physical Systems (CPSs). Finally, (Li et al., 2022) explored the use of DNN for accurate classification and analysis of IoT smart cities data, achieving a prediction accuracy of 97.80%.

Several studies have investigated the use of DNN on resource-constrained mobile devices for device-level applications. For instance, (Tang et al., 2017) conducted an investigation into the suitability of a compiler-based platform for benchmarking DNN inference on mobile devices. Meanwhile, (Iandola and Keutzer, 2017) focused on minimizing the computational resources required for deploying DNN in such environments by proposing various procedures for creating a smaller DNN architecture. However, their work lacked empirical evaluations. In contrast, (Shen et al., 2020) proposed a technique for compressing CNN to enable structure learning in IoT resource-constrained environments. Their approach demonstrated promise on benchmark datasets such as CIFAR-10 and Imagenet, but failed to consider memory usage and lacked evaluation on IoT benchmark datasets. Similarly, (Kodali et al., 2017) utilized FCNN for classification tasks on resource-limited devices. However, their approach may not be scalable for constrained IoT devices due to the lack of consideration for model complexity during FCNN architecture selection.

Our paper proposes a novel approach that targets effective attack detection with resource minimization by reducing FCNN computational complexity. The method employs pruning, simulated micro-batching, and parameter optimization to regularize the resulting DNN model and reduce memory and time requirements while increasing accuracy performance. This approach distinguishes itself from existing proposals in the literature, which typically compress DNN by quantizing weights and bias parameters. Overall, this study aims to reduce computational complexity while enhancing the accuracy of FCNN-based models for effective attack detection in resource-constrained environments.

2.2. Adversarial attacks against AI

Adversarial attacks can significantly degrade the performance of AI-based models used in IoT security monitoring by exploiting vulnerabilities in the model. One such attack is the data poisoning attack, where an attacker modifies the training data by injecting poisonous instances to manipulate the model's learning process (Shafahi et al., 2018). This can cause the model to misclassify legitimate instances, resulting in compromised security (Pitropakis et al., 2019). In addition to poisoning attacks, perturbation-based attacks such as Fast Gradient Sign Method (FGSM), and Projected Gradient Descent (PGD) (Kurakin et al., 2016), semantic (Hosseini et al., 2017), and random noise (Athalye et al., 2018) attacks can be used to generate new adversarial samples during the testing phase.

The aforementioned perturbation methods employ a white-box approach, assuming the adversary has full knowledge of the cybersecurity monitoring model. Therefore, they are commonly used in IoT security monitoring (Aloraini et al., 2022). Pujari et al. (2022) investigates the effectiveness of conventional ML-based models against adversarial attacks crafted with IoT network security datasets, while (Abou Khamis and Matrawy, 2020) examine the robustness of DNN in similar scenarios, but only considering adversarial attacks generated using IP-based datasets. However, a robust and efficient classification model can withstand a wide range of adversarial perturbations achieved by training the model with perturbed samples to enhance regularization for resilience testing (Tramèr et al., 2017). This paper proposes an approach to counter IoT security attacks without using perturbed samples during training. The goal is to leverage the optimized REDNN model's ability to combat adversarial attacks effectively and efficiently. Furthermore, we investigate the impact of implementing 16 bit Full Precision (FP16) on the FCNN and REDNN model's robustness to evaluate the feasibility of using a lightweight and robust DNN model in a resource-constrained IoT environment.

Algorithm 1 FCNN training.

Input: Labelled data $\mathcal{T}_d$, Number of iteration $\mathcal{T}$, Batch size S
Output: Baseline model $\mathcal{M}_b$

1: **function** BASE($\mathcal{T}_d[\,]$) ▷ Training baseline model
2: **for** $i = 1$ to $\mathcal{T}$; **do**
3: Mini-batch $B = \{(x_1, y_1), ..., (x_m, y_m\} \subset \mathcal{T}_d$ ▷ Size S
4: $F_p(B)$ ▷ Forward propagation
5: $\mathcal{E}_i \leftarrow L$ ▷ L = Base loss
6: B_p(B) ▷ Backward propagation
7: Compute gradients for parameters update
8: Estimate m_i ▷ Execution memory at epoch i
9: Estimate t_i ▷ Execution time at epoch i
10: $\mathcal{M}_b$ = Trained model that estimate $\mathcal{E}_i, t_i, m_i$
11: **end for**
12: **return** $(\mathcal{M}_b, t_i, m_i, \mathcal{E}_i)$
13: **end function**

2.3. Federated learning (FL) in IoT environment

In the domain of IoT security monitoring, FL is gaining popularity. Preuveneers et al. (2018) investigated FL applications for intrusion detection in IoT networks, while (Lim et al., 2020) and (Imteaj et al., 2021) identified open research problems on FL for resource-constrained IoT devices. Additionally, (Nguyen et al., 2019) proposed a signature-based FL approach to detect attacks on IoT devices, and (Liu et al., 2020) leveraged FL capabilities to detect attacks on Industrial IoT (IIoT) devices by training a DNN model in a federated manner using a labeled dataset. The authors integrated CNN and Long Short-Term Memory (LSTM) for better model convergence. However, the MNIST and CIFAR-10 datasets utilized for estimating the model parameter gradients are non-IoT data. In contrast, (Jiang et al., 2019) employed model pruning for efficient FL training on edge devices, utilizing an image dataset. Meanwhile, (Bonawitz et al., 2019) proposed a TensorFlow-based FL framework for mobile devices, utilizing Android mobile devices for evaluation. Additionally, (Popoola et al., 2021) utilized FL to detect a zero-day attack in an IoT network environment, taking advantage of FL data privacy without considering resource limitations. In their investigations, they used the N-BaIoT (Meidan et al., 2018) device-centric dataset.

However, existing proposals in the literature do not take into account the optimization of FL training specifically for reducing memory consumption on IoT networks. This paper addresses this limitation in Section 3.4 by optimizing the federated training process using techniques such as pruning, micro-batching, and parameter regularization, which are specifically tailored to enhance resource efficiency in the context of FL training on IoT networks.

3. Methodology

To validate the viability of the proposed approach, an evaluation was conducted using a FCNN on multiple IoT benchmark datasets. The optimization algorithm of the FCNN was leveraged to generate the REDNN.

3.1. Fully connected neural network (FCNN)

A FCNN is a type of neural network consisting of multiple layers of neurons that process input data. Each neuron computes an output based on its activation function and input values, and these neurons are connected in a non-linear pattern of layers using weights and bias parameters. The weights and biases serve as information storage units and control the flow of operations within the network. In this study, we used Algorithm 1 to obtain the optimized FCNN model ($\mathcal{M}_b$) as a baseline for comparison. The BASE function in line 1 of Algorithm 1 corresponds to

Algorithm 2 Proposed algorithm to obtain REDNN.

Input: Penalty term λ, ($\mathcal{T}_d$, $\mathcal{T}$ and B in Algorithm 1)
Output: Efficient model $\mathcal{M}_e$

1: **function** EFFICIENT($\mathcal{T}_d[\,]$)
2: **for** $j = 1$ to $\mathcal{T}$; **do**
3: Micro-batch $M = \{(x_1, y_1), ..., (x_m, y_m)\} \subset B$
4: $F_p(M)$ ▷ Forward propagation
5: $\mathcal{E}_t = L$ ▷ L = Initial loss
6: m_t, t_t ▷ m_t, t_t estimated memory and time using $\mathcal{E}_t$
7: $\mathcal{E}_j \leftarrow \mathcal{E}_t + \lambda \sum_{j=1}^{W} \frac{(w_j^2/w_0^2)}{(1+w_j^2/w_0^2)}$
8: B_p(M) ▷ Backward propagation
9: Compute gradients for parameters update
10: **if** ($\mathcal{E}_j \leq \mathcal{E}_t$) **then**
11: $\lambda = \lambda + \triangle\lambda$
12: Estimate m_j ▷ Execution memory at epoch j
13: Estimate t_j ▷ Execution time at epoch j
14: **if** (($m_j < m_t$) $\wedge$ ($t_j < t_t$)) **then**
15: $m_t = m_j$ ▷ m_t = Efficient memory
16: $t_t = t_j$ ▷ t_t = Efficient time
17: $\mathcal{M}_e$ = Trained model that estimate $\mathcal{E}_j, m_t, t_t$
18: **end if**
19: **end if**
20: **end for**
21: **return** ($\mathcal{M}_e, \mathcal{E}_j, m_t, t_t$)
22: **end function**

mini-batch training with the gradient descent algorithm, which minimizes the objective loss function (L) in Equation (1), specifically the negative log-likelihood (cross-entropy), to learn from the training set ($\mathcal{T}_d$) and map unseen samples. The resulting FCNN approach is a supervised neural network classifier, $\mathcal{M}_b$, which takes an input $\mathcal{T}_d$ and outputs a probability class vector $\hat{Y}$. The desired output $\hat{Y}$ is rounded to the closest integer using a specified threshold value t as in Equation (2), representing either a benign (1) or an attack (0) traffic instance.

$$L = \frac{1}{m}\sum_{i=1}^{m} -(Y_i * \log(\hat{Y}_i) + (1 - Y_i) * \log(1 - (\hat{Y}_i)) \tag{1}$$

$$Detection = \begin{cases} 0 \text{ if } \hat{Y} \leq t \\ 1 \text{ if } \hat{Y} > t \end{cases} \tag{2}$$

3.2. Robust effective and resource efficient DNN (REDNN)

As highlighted in section 3.1, training a resource-efficient DNN model can be an intricate task, particularly in the context of IoT security monitoring (Abiodun et al., 2018). The intricacy of this task is due to the numerous rounds of training iterations and the requisite DNN model parameters needed to design and build an optimal network architecture. This complexity is compounded when building an efficient threat detection system with supervised DNN for cybersecurity monitoring, especially when dealing with multidimensional datasets. To address this issue, the baseline model $\mathcal{M}_b$ in Algorithm 1 is used to obtain its resource-efficient counterpart (REDNN). The training procedure, in Algorithm 2, optimizes a function using $\mathcal{T}_d$ to obtain the efficient M_e equivalent to the REDNN model. To achieve this, the optimization procedure utilizes micro-batching (Oyama et al., 2018; Huang et al., 2019) for efficient model training that is suitable for on-device learning as well.

The function procedure requires $\mathcal{T}_d$ in mini-batch and micro-batch forms and iterates $\mathcal{T}$ times repeatedly to return the efficient M_e representing the REDNN model. The optimization process utilizes a penalty function (weight elimination) (Han et al., 2015) represented by E in Equation (3) with a weight threshold parameter w_0. The expression in line 7 of Algorithm 2 is responsible for pruning the network model weights to reduce its architectural complexity. This procedure is useful in distinguishing the sets of relevant weights that can enable efficient model learning from the irrelevant ones, particularly the insignificant large weights of the baseline $\mathcal{M}_b$ model.

In the process, weight values W greater than w_0 can yield a complexity cost closer to 1 and require regularization using the penalty parameter λ. This is important to reduce the complexity of the model to enable faster training. As we are more concerned with a less complex, efficient, and effective model building that can retain its performance, we consider the set of parameters that can give a training error $\mathcal{E}_j$ lower than $\mathcal{E}_t$. The most important parameters are the w_0 and λ, which are the threshold that controls the learning of the model while reducing its architecture.

In line 10 of Algorithm 2, the regularization error $\mathcal{E}_j$ is compared with the initialized error $\mathcal{E}_t$ before regularization. This is to examine the convergence rate of the model during each epoch iteration. Based on the outcomes of line 10, relaxation of the λ value using the $\triangle\lambda$ occurs in line 11. After these steps, the memory footprint and execution time are estimated in lines 12 and 13 and compared with the initialized values from line 6 in line 14. This process aims to find a model architecture with a faster convergence rate and minimal memory and time requirements for training and testing. Due to the regularization in lines 10 and 11 of Algorithm 2, the returned REDNN model is less complex.

$$E = \lambda \sum_{j=1}^{W} \frac{(w_j^2/w_0^2)}{(1 + w_j^2/w_0^2)} \tag{3}$$

3.3. Federated deep neural network (BFDNN)

Given the widespread adoption of FL, particularly in cybersecurity monitoring, its exploration in the context of IoT security monitoring can be beneficial. FL's ability to preserve on-device training data can be useful in proposing AI-based security mechanisms for resource-constrained IoT devices. To this end, the Baseline Federated Deep Neural Network (BFDNN) training procedure is illustrated in Fig. 1, which utilizes the function BASE to train a baseline model using stochastic gradient descent in FL settings. Algorithm 3 describes the details of this procedure, where each client performs iterative rounds of gradient descent weights for model aggregation, and Device UPDATE distributes a master model to each client's subset at each communication round. Using the pro-

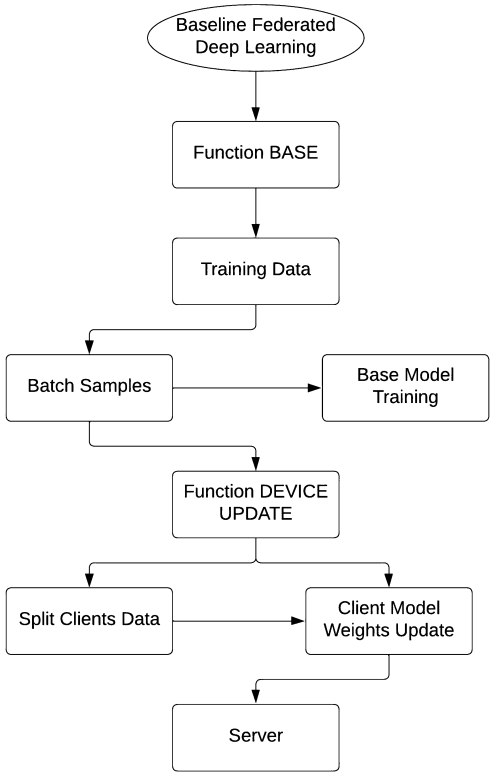


Fig. 1. Baseline federated learning procedure.

Algorithm 3 BFDNN training on each distributed node.

Input: Labelled data $\mathcal{D}_{tr}$, Iteration number $\mathcal{T}$, Batch size $\mathcal{S}$
Output: Baseline federated model $\mathcal{M}_n$

1: **function** BASE($\mathcal{D}_{tr}$[ ])
2: **for** $i = 1$ to $\mathcal{T}$ **do** ▷ for each local epoch iterations
3: Mini-batch $B = \{(x_1, y_1), ..., (x_m, y_m)\} \subset D_{tr}$ ▷ Mini-batch size $S_m \leftarrow |D_{tr}|//S$
4: $F_p(B)$ ▷ Forward propagation with B
5: $\mathcal{E}_i \leftarrow L$ ▷ L = Base loss
6: B_p(B) ▷ Backward propagation
7: **function** DEVICE UPDATE((d, w)) ▷ Run on device d with weights w
8: $B_s \leftarrow$ (Split data B into batches of size S_m) ▷ S_m is a local Mini-batch size
9: **for** batch $b \in B_s$ **do**
10: $w \leftarrow$ local weights update ▷ device local weights update computation
11: Estimate m_i ▷ Execution memory at epoch i
12: Estimate t_i ▷ Execution time at epoch i
13: $\mathcal{M}_n$ = Trained model that estimate $\mathcal{E}_i, m_i, t_i$
14: **end for**
15: **end function**
16: **end for**
17: **return** w to server in Algorithm 5 ▷ Calls to coordinating server in Algorithm 5 for weights averaging
18: **return** $(\mathcal{M}_n, \mathcal{E}_i, m_i, t_i)$
19: **end function**

Algorithm 4 Proposed REFDNN training on each distributed node.

Input: Penalty term λ, ($\mathcal{D}_{tr}, \mathcal{T}, B, L$ and S_m in Algorithm 3)
Output: Efficient federated model $\mathcal{M}_f$

1: **function** EFFICIENT($\mathcal{D}_{tr}$[ ])
2: **for** $j = 1$ to $\mathcal{T}$; **do**
3: Micro-batch $M = \{(x_1, y_1), ..., (x_m, y_m)\} \subset B$ ▷ $B \subset \mathcal{D}_{tr}$
4: $F_p(M)$ ▷ Forward propagation with M
5: $\mathcal{E}_t = L$ ▷ Initialized loss
6: Estimate m_t, t_t Initialized memory and time based on $\mathcal{E}_t$
7: $\mathcal{E}_j \leftarrow \mathcal{E}_t + \lambda \sum_{j=1}^{W} \frac{(w_j^2/w_0^2)}{(1+w_j^2/w_0^2)}$
8: B_p(M) ▷ Backward propagation with M
9: **function** DEVICE UPDATE((d)) ▷ Run on device d
10: $M_s \leftarrow$ (Split data M into batches of size S_m)
11: **for** batch $b \in M_s$ **do**
12: $w \leftarrow$ local weights update ▷ device local weights update computation
13: **if** $(\mathcal{E}_j \leq \mathcal{E}_t)$ **then**
14: $\lambda = \lambda + \triangle\lambda$
15: Estimate m_j ▷ Execution memory at epoch j
16: Estimate t_j ▷ Execution time at epoch j
17: **if** $((m_j < m_t) \wedge (t_j < t_t))$ **then**
18: $m_t = m_j$ ▷ m_t = Efficient memory
19: $t_t = t_j$ ▷ t_t = Efficient time
20: $\mathcal{M}_f$ = Trained model that estimate $\mathcal{E}_j, m_t, t_t$
21: **end if**
22: **end if**
23: **end for**
24: **end function**
25: **end for**
26: **return** w to server in Algorithm 5 ▷ Calls to Algorithm 5 for model weights averaging
27: **return** $(\mathcal{M}_f, \mathcal{E}_j, m_t, t_t)$
28: **end function**

posed method in Algorithm 4, resource efficient version of this standard FL approach was obtained.

3.4. Resource efficient federated deep neural network (REFDNN)

Training a resource-efficient DNN model for FL tasks can be a challenging task, especially in an IoT network environment. This is due to the FL communication rounds and DNN model parameters requirements in building the desirable network architecture (He et al., 2022). The complexity of such an approach increases with multidimensional datasets. A FedAvg core model (BFDNN) was examined with FCNN and CNN model variations against some IoT and non-IoT benchmark datasets and its optimization algorithm was exploited to obtain REFDNN. This optimized training procedure is illustrated in Algorithm 4. For better performance, the set of model parameters that can produce a lower error based on line 7 of Algorithm 4 was utilized. The function procedure in Algorithm 4 is responsible for computing

Algorithm 5 Coordination Procedure for Algorithm 3 and 4.

Server Executes:

1: **function** SERVER WEIGHTS UPDATE
2: initialize weight w_0;
3: initialized $j = 1$
4: **while** $j \leq f$ **do** ▷ f is the number of federated round
5: $m \leftarrow max(C.K, 1)$ ▷ $C.K$ fraction of clients K
6: $R \leftarrow$ random set of S_j ▷ $S_j \leftarrow$ random set of f clients
7: **for** $k \in R$ in parallel **do** ▷ k client index, a selected clients from R
8: Weight update for each client k ▷ Federated model weight update for Algorithm 3 or 4
9: **end for**
10: Averaged weights update ▷ Average weights update based on client K weights
11: $w_{j+1} \leftarrow \sum_{k=1}^{K} \frac{f_k}{f} w_{j+1}^k$ ▷ f_K = client k sample size, f total sample size
12: $j = j + 1$
13: **end while**
14: **return** Averaged updated weights
15: **end function**

and updating client device weights at each local epoch iteration before sending them to the coordinating server. In line 13 of Algorithm 4, the device model error is compared with the initialized error before model regularization in line 14. Following this stage, computational memory footprints and execution time were estimated in lines 15 and 16. Subsequently, in line 17, these estimates were compared to the initialized values mentioned in line 6 in order to determine the minimal memory constraint generated by the client device model. Device models with minimal resource consumption are returned to the coordinating server in Algorithm 5 together with their weights for model averaging. Then, the coordinating server can update the client model weights in a federated setting and perform weight averaging while returning the updated averaged weights for model aggregation. This process can reduce the client's communication time and computational complexity while building the resource-efficient aggregate model of REFDNN. The memory and execution time savings for each client device at each federated round and accumulating all these savings can lead to significant savings when the model is converged.

4. Evaluation

This section outlines the evaluation criteria for the FCNN and REDNN models and provides information on the datasets used to create them. The datasets used in this study include N-BaIoT (Meidan et al., 2018), Kitsune (Mirsky et al., 2018), and WUSTL (Teixeira et al., 2018), each of which is briefly described.

4.1. Utilized datasets

The N-BaIoT dataset comprises authentic data samples obtained from nine commercial IoT devices that demonstrate various botnet and benign network traffic flows (Meidan et al., 2018). These devices include (i) Danmini Doorbell, (ii) Ecobee Thermostat, (iii) Ennio Doorbell, (iv) Philips B120N10, (v) Provision PT-737E, (vi) Provision PT-838, (vii) Samsung SNH-1011-N, (viii) SimpleHome XCS-1002-WHT, and (ix) SimpleHome XCS-1003-WHT. These devices have either been affected by BASHLITE or Mirai attacks, or have been operating normally. Each device has extensive records of attacks and regular instances that comprise 115 feature vectors. Consequently, the N-BaIoT dataset is an ideal benchmark for developing IoT network intrusion detection systems. The FCNN and REDNN models were trained and tested utilizing device data from N-BaIoT.

The Kitsune dataset contains various network traffic captured in an IoT setting (Mirsky et al., 2018). The dataset comprises attacks that breach confidentiality, integrity, and authenticity, and these attacks are categorized into (i) reconnaissance attacks, (ii) DoS attacks, and (iii) Mirai attacks. The subset of the dataset used to evaluate our models comprises 764,137 instances of Mirai and normal traffic. The dataset has 115 features and a normal distribution of 121,621 raw traffic data.

Table 2
Topology and distribution of normal and attack for each device data.

Device	Normal	Attack	Inputs	Output	Topology
Danmini Doorbell	49,548	968,750	115	1	128-128-128-128
Ecobee Thermostat	13,113	822,763	115	1	32-64-64-16
Ennio Doorbell	39,100	316,400	115	1	64-128-128-64
Philips B120N10	175,240	923,437	115	1	128-128-128-128
Provision PT-737E	62,154	766,106	115	1	128-128-128-128
Provision PT-838	98,514	729,862	115	1	128-128-128-128
Samsung SNH-1011-N	52,150	323,072	115	1	128-128-128-128
SH XCS-1002-WHT	46,585	816,471	115	1	128-128-128-128
SH XCS-1003-WHT	19,528	831,298	115	1	128-128-128-128
Kitsune	121,621	642,516	115	1	128-128-128-128
Wustl	6,566,438	471,545	6	1	128-128-128-128

WUSTL dataset consists of multiple reconnaissance attacks with normal traffic that emulate real-world industrial IoT systems for CPSs security research (Teixeira et al., 2018). This dataset is useful for investigating the feasibility of ML algorithms for detecting various real-world attacks. The raw data consists of 7,037,983 data samples with seven (7) features. It comprised 93.30% benign records with 6.70% attacks data records.

4.2. Data preprocessing

The datasets were carefully selected for frequent model training and thorough evaluation. These datasets provide numerical traffic flow information, which we utilized in our investigations. Each dataset was split into training and testing samples, with 80% allocated for training and 20% for testing purposes. The data input vectors underwent normalization using unity-based normalization and feature scaling. In a dataset comprising n data features, namely $x_1, x_2, ..., x_n$, normalization was performed using the formula specified in Equation (4). The normalized value of the i^{th} feature is denoted by x_i', while x_i represents its original value. Additionally, min_{x_i} and max_{x_i} represent the minimum and maximum values of the i^{th} feature across the entire dataset, respectively.

$$x_i' = \frac{x_i - min_{x_i}}{max_{x_i} - min_{x_i}} \tag{4}$$

Furthermore, in addition to the datasets employed in section 4.1 and the preprocessing procedures outlined in section 4.2 for technique implementation, we incorporated the MNIST dataset into our study. The MNIST dataset is an appropriate benchmark for evaluating the model's learning capacity over non-IoT cybersecurity datasets. This evaluation is crucial to investigate whether the proposed model can efficiently detect complex patterns in other datasets. The MNIST handwritten digits dataset is a subset of the dataset from the National Institute of Standards and Technology (Baldominos et al., 2019). It comprises 60,000 training digit samples and 10,000 testing digits, which are size-normalized and consist of 28*28 images with 256 gray levels.

4.3. Experimental setup

We employed Python version 3.76 to construct each model on a desktop computer with Intel Xeon E5-2695 CPUs, containing 4 cores and running at 2.10 GHz, with 16.0 GB of installed memory. We utilized the integrated memory usage to profile the model's memory consumption (Pedregosa and Gervais, 2019). During training, the parameters remain constant to ensure a fair comparison. This practice is applied to the baseline FCNN model, the optimized REDNN model, and the adversarial process.

4.4. Implementation details

4.4.1. FCNN and REDNN models design

To build the generic sequential (dense) FCNN and REDNN models for each dataset, we employed the scientific NumPy Python module (Johansson, 2018). This module enables the creation of a comprehensive DNN model without any library, providing insights into the underlying concepts and internal operations within the network. Each model consists of an input layer, four hidden layers, and an output layer, as shown in Table 2. To determine the topology selection for each dataset, we utilized the best-run Hyperas modules (Komer et al., 2019). This allowed us to choose the most optimal topology configurations for each dataset, which minimizes operations while maximizing performance metrics. These requirements are essential for binary classification tasks. The architectural settings remain consistent for evaluating both the baseline FCNN and the proposed REDNN model. Table 2 provides a detailed description of the model topology for each tested dataset.

During training, a mini-batch gradient descent optimizer with momentum was utilized. The weight and bias parameters were randomly initialized within the range of $[0, 1]$. For both the baseline and optimized training procedures, a learning rate of $lr = 0.001$ was used across each dataset, except for the Ecobee and Ennio devices data, which had a different topology and used a learning rate of $lr = 0.0001$. Both FCNN and REDNN models used a momentum value of 0.001. The REDNN model was built using 4 micro-batches, with values of 0.01 for λ, $\triangle\lambda$, and threshold ω_0 (Bosman et al., 2018). The models were trained with 128 batches within 100 epochs for accuracy to converge. The loss function was calculated using binary cross-entropy, with ReLu (Ide and Kurita, 2017) used as the activation function in the input layer and Sigmoid for the output layer. To efficiently select hyperparameters, an automatic optimizer search module (Pumperla, 2018) was employed. This technique required a range of values for each hyperparameter to be tuned to return an efficient combination. The Numpy.float16 module was used to implement FP16 for the baseline and optimized models.

We employed TensorFlow Core version (v2.8.0) (David et al., 2021) to build Keras and TensorFlow DNN models. The TensorFlow Lite (TFLite) converter module is used to create the TFLite DNN model. To ensure a fair comparison, Numpy (FCNN and REDNN) is also used, and both the Keras and TFLite models are trained in 128 mini-batches using stochastic gradient descent, at 100 epochs iterations. Scikit-learn (Pedregosa et al., 2011) ML python framework is used for the linear SVM, Adaboost, and GB models. The study codes are publicly available at (Zakariyya, 2021) for exploration and reproduction purposes. The GitHub repository (Zakariyya, 2021) includes both the Jupyter notebook file and the Python script for the TFLite experimentation.

4.4.2. Adversarial attacks implementation

To generate adversarial samples of FGSM and PGD (Kurakin et al., 2016), we utilized Equation (6) along with the cleverhans documentation (Papernot et al., 2020). The FGSM involves a one-step gradient update towards the direction of the gradient sign (see Equation (5)).

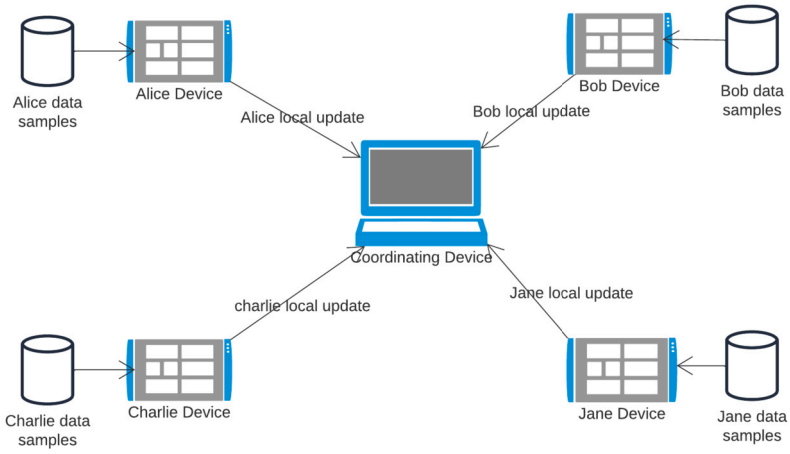


Fig. 2. BFDNN and REFDNN model training testbed with gigabyte devices.

The notation X^o represents the original data, ϵ represents the adjustment step of the original data, Y is the label, θ represents the model parameters, ∇X^o is the backward propagation step for gradient update, and $J(l, X^o, Y))$ is the loss function used to train the network.

In the PGD attacking method, an initialized noise $\mathcal{U}(-\epsilon, \epsilon)$ based on the uniform distribution of ϵ is added to the original data sample before generating and clipping the adversarial samples repeatedly. Then, Equation (6) is iterated t times to generate the perturbed samples, where $\Pi X^o + S$ represents the projection of the perturbation set $X^o + S$ using the projection operator Π, α is the gradient step size, and J is the loss function.

$$X^{fgsm} = X^o + \epsilon * sign(\nabla_{X^o} J(\theta, X^o, Y)) \tag{5}$$

$$X^p = X^{t+1} = \Pi_{X^o+S}(X^t + \alpha * sign((\nabla_{X^o} J(\theta, X^o, Y))) \tag{6}$$

The semantic attack (Hosseini et al., 2017) method was generated by inverting the normalized data $X = x_i, i = 1, 2, ...n$ within $[0, 1]$. For random noise, the noise data are generated based on the uniform distribution of the original data. For normalized data within $[0, 1]$, the introduced noise will be in the form of $\mathcal{U}(0, 1)$.

Another perturbation procedure considered in this paper is data poisoning attacks described in Algorithm 6. In this scenario, the data is poisoned by randomly flipping the labels (based on a random split of data features). The flipping procedure considers label modification for attack (0s) and benign (1s) samples. This is the all-label modification technique that changes 1s to 0s and 0s to 1s, respectively. It is a non-targeted form of adversarial attack method that concentrates on both the benign and attack traffic classes. The rationale is to mislead the model by lowering its accuracy value to make it a weaker model. It achieved this by injecting modified labels for each data feature while training the model. The trained model used testing data with correctly assigned labels for validation. We generate this form of attack by considering the training dataset. During implementation, the data samples are randomized before splitting to have a fair proportion of attack and benign samples. All labels of the randomized samples are flipped based on the specified poisoning proportion, and to increase the chance of the success rate, we consider the rate to be from 0%–50% by 5% increment. Each tested perturbation method used the preprocessed datasets described in section 4.1. These datasets are used to examine the success rate of each perturbation method to investigate REDNN resilience.

4.4.3. Virtual workers FL setup

The virtual on-device training utilized PyTorch version 1.4.0 (Paszke et al., 2019) and PySyft version 0.2.9 (Ryffel et al., 2018) frameworks. The PySyft framework simplified the creation of virtual workers, which were used to simulate the FL scenario for the BFDNN and the proposed REFDNN. These workers emulate real virtual machines and can run as a separate process within the same Python program with their dataset. The federation training procedure considered four clients virtual workers and a coordinating server worker receiving computational updates from each virtual client worker model. Each federated client model comprised an input layer, four hidden layers, and an output layer. The topology selection against each dataset utilized the method proposed by (Komer et al., 2019) to minimize operations and improve performance metrics. The experimental settings were appropriate for binary classification, as indicated by the parameter tuning technique employed by (Komer et al., 2019). The overall architectural settings remained identical for evaluating the BFDNN and the proposed REFDNN technique.

Algorithm 6 Label modification perturbation procedure.

Input: $\mathcal{X}, \mathcal{Y}, n, p =$ data, labels, data length, percent
Output: Poisoned data $\{\mathcal{X}', \mathcal{Y}'\}$

for $t = 1$ *to* n; **do**
  if $t \in (1, p * n)$ **then** ▷ Random samples selection as the dataset was randomized
    $y_t = 1 - y_t$ ▷ Labels 0 and 1 modification
    $\mathcal{Y}' = \{(x_t, y_t)\}, t = 1 \ldots n$ ▷ Integrating labels
  end if
end for
return $\{\mathcal{X}', \mathcal{Y}'\}$

4.4.4. Testbed FL setup

In order to assess the efficient federated communication of the REFDNN against BFDNN in a testbed setting, we utilized the PySyft version 0.2.9 (Ryffel et al., 2018) python framework over a network (see Fig. 2 with a client and server-class connected via a WebSocket (WS). As PyTorch is a compatible library for PySyft, we employed it to develop an edge computing FL training scenario suitable for resource-constrained devices. The environmental settings replicated the client-server communication scenario in a distributed manner, thereby enabling the creation of realistic testbed settings. To build this network, we employ 4 Gigabyte Brix (GB-BXBT-2807) mini PCs and a laptop as shown in Fig. 2. The personal laptop served as the coordinating server in a wireless network, emulating low-frequency connections. The server was responsible for aggregating and distributing model weights to clients. The client devices in Algorithms 3 and 4 were responsible for locally training the model using the server model weights on the client's dataset and returning client weights to the server. Therefore, the communication workload was higher at the client-side containing the edge devices than the server machine. The installed Operating System (OS) on GB-BXBT-2807 clients was Ubuntu version 20.04.4 LTS. Each client contained an installation of the PySyft framework and its dependencies. The Federated network testbed implementation codes are publicly accessible (Zakariyya, 2022).

To assess the simulated runtime and real execution time of both BFDNN and REFDNN, we conducted experiments involving four work-

Table 3
Testing memory footprint (cumulative).

Dataset	Model	Mem (MB)	Mem save (%)	Test acc (%)
Danmini Doorbell	FCNN	3.742	N/A	95.11
	REDNN	1.555	58.44	95.11
Ecobee Thermostat	FCNN	2.804	N/A	93.36
	REDNN	1.277	54.46	93.36
Ennio Doorbell	FCNN	2.410	N/A	88.94
	REDNN	0.539	77.63	88.94
Philips B120N10	FCNN	3.738	N/A	84.08
	REDNN	1.731	53.71	84.08
Provision PT-838	FCNN	3.031	N/A	88.07
	REDNN	1.266	58.23	88.07
Provision PT-737E	FCNN	3.008	N/A	92.52
	REDNN	1.285	57.28	92.52
Samsung SNH-1011-N	FCNN	2.598	N/A	86.07
	REDNN	0.582	77.60	86.07
SH XCS-1002	FCNN	3.004	N/A	94.65
	REDNN	1.320	56.06	94.65
SH XCS-1003	FCNN	3.145	N/A	97.72
	REDNN	1.305	58.51	97.72
Kitsune	FCNN	2.726	N/A	84.09
	REDNN	1.168	57.15	84.09
Wustl	FCNN	491.6	N/A	94.26
	REDNN	5.711	98.84	94.26

ers (Alice, Bob, Charlie, and Jane, as illustrated in Fig. 2), each with their distributed training data. To ensure optimal model convergence, we employed a federated communication round consisting of 50 iterations with two epochs, using a mini-batch size of 64. We selected a test batch sample size of 1000 with a learning rate of 0.01 to facilitate effective FedAvg SGD training. The real-time models used for each federated client in Algorithm 3 and 4 featured an input layer and four identical hidden layers (128-128-128-128), along with an output layer, as appropriate. This architecture was chosen to promote efficient and effective model convergence.

To evaluate the effectiveness and generalizability of REFDNN, we also implemented a CNN DNN variant in realistic settings, with clients using the MNIST image dataset (Deng, 2012). This CNN architecture comprised two convolutional layers (Conv-2D). The first 2D convolutional layer required one input to output 20 convolutional features, using a 5 square kernel (1, 20, 5, 1). The second 2D convolutional layer required 20 input layers to output 50 convolutional features, using a 5 square kernel (20, 50, 5, 1). The architecture in the first real-time layer was (800 (4*4*50), 128), with (128, 10) in the second real-time layer. Max-Pool in 2D was run over the input image without dropout utilization. The fully connected hidden layers in the convolutional architecture were similar to the version described in Table 2.

5. Results and discussion

This section presents an overview of the experimental results. It provides an in-depth evaluation comparison between the REDNN and optimized FCNN models, with a focus on resource efficiency, effectiveness, and adversarial robustness across datasets. Furthermore, it elaborates on the evaluation comparison between REFDNN and BFDNN federated models in an IoT environment.

5.1. REDNN model effectiveness and resource efficiency

To assess the effectiveness and resource efficiency of the models, Table 3 presents the measured testing results of eleven IoT datasets run with both the FCNN and REDNN models. In each case, the models' testing memory footprint is profiled in megabytes (MB). As anticipated, the REDNN model demonstrated a non-accuracy degradation performance while consuming a minimal memory footprint. Specifically, it can process the Wustl and Ennio Doorbell datasets with 98.84% and 77.63% memory savings, respectively, compared to the baseline FCNN model. These resource optimizations position the REDNN model as a preferred option for IoT security monitoring, as they suggest the potential to reduce computational resources without compromising accuracy. Additionally, the findings indicate that deploying the model in a resource-constrained environment is feasible.

Table 4
Training performance evaluation with testing accuracy across frameworks with Provision PT-737E dataset (per record).

Procedure	Train time (ms)	Train mem (B)	Test set acc (%)
FCNN-Keras	13.189	3127.5	92.52
FCNN-TFLite	0.1605	372.29	92.52
FCNN-Numpy	0.0571	16.933	92.52
REDNN-Numpy	0.0196	0.1388	92.52

Table 4 provides a detailed comparison of REDNN's performance evaluation as implemented in various state-of-the-art technology frameworks (libraries). This comparison highlights the potential of REDNN in saving resources across different experimental platforms. During training, REDNN demonstrates efficient performance with better memory footprint and time savings for each data record. Specifically, it saves 99.85% and 99.99% of training time and memory footprint, respectively, compared to the baseline model trained with Keras, as computed based on the reported values in columns Train time $(((0.0196/13.189) * 100) - 100)$ and Train mem $(((0.1388/3127.5) * 100) - 100)$ from Table 4. In comparison with the converted FCNN TFLite model, REDNN exhibits better memory usage. This could be attributed to the fact that the TFLite model inherits the default Keras parameters during model conversion, resulting in a lighter version of the Keras model. However, the quantized optimized TFLite model consumes fewer resources, requiring 0.010 ms of training execution time and 0.0060 B of training memory footprint. It is worth noting that the use of low precision in some cases can lead to numerical issues, causing a degradation in accuracy performance with certain datasets. Therefore, we implement each framework in 32 bits and compare their performance in Table 4 to investigate resource savings without low precision integration. The significant training resource-saving of the optimized REDNN model could be beneficial for on-device learning. These compelling results provide a strong basis for utilizing the optimized REDNN model to evaluate the hypothesis stated in RQ1.

Table 5 presents the testing resources consumed by each model using different technology frameworks. The table shows that the NumPy implementation is the fastest among the tested frameworks. Additionally, REDNN demonstrates more efficient processing of IoT data than the baseline FCNN model when run in the same framework. The TFLite model is more efficient than the Keras model but slower than the Numpy (FCNN and REDNN) models. Interestingly, REDNN outperforms the other models in terms of processing time savings, achieving savings of 4.31%, 69.81%, and 80.55% compared to the FCNN, TFLite, and Keras models, respectively. These results demonstrate the resource-efficient nature of our training procedure using Numpy and suggest that it can be an appropriate method for training and building effective models in a resource-constrained environment, outperforming the currently available state-of-the-art methods.

Regarding memory consumption in column (Test mem), REDNN demonstrates better savings with each data record. For FCNN-Numpy, FCNN-TFLite, and FCNN-Keras models, the memory footprint was reduced by 78.91%, 80.12%, and 98.51%, respectively. The TFLite's higher resource consumption is due to the data type conversion during prediction (TensorFlow, 2022). The conversion can increase the

Table 5
Testing resource consumption across frameworks with Provision PT-737E dataset (per record).

Procedure	Test time (ms)	Test mem (B)	Test set acc (%)
FCNN-Keras	2.3522	512.64	92.52
FCNN-TFLite	1.5155	38.533	92.52
FCNN-Numpy	0.4781	36.317	92.52
REDNN-Numpy	0.4575	7.6606	92.52

Table 6
Performance evaluation comparison on Provision PT-737E dataset (per record).

Model	Train time (ms)	Test time (ms)	Train mem (B)	Test mem (B)	Test set acc (%)
SVM	909.64	500.87	378.96	923.48	92.52
GB	32.621	0.2242	22.230	20.018	92.58
AdaBoost	31.212	2.6126	4.1910	13.842	92.47
FCNN	0.0571	0.4781	4.2333	7.9685	92.52
REDNN	0.0196	0.4575	0.0347	7.6606	92.52

execution time and memory (intel, 2020) as demonstrated in Table 5. The higher resource (memory and time) consumption of the TFLite at the testing stage is a limitation for effective IoT attack detection. The REDNN algorithm's minimal resource consumption suggests its potential efficacy as a mechanism for IoT security monitoring, as well as for the security monitoring of other cyber-physical devices. Notably, resource-efficient ML plays a crucial role in IoT security monitoring for a variety of reasons. For example, as mentioned above, IoT devices often operate with limited resources, such as memory, processing power, and battery life, and, thus, optimized algorithms such as REDNN can be deployed on such devices without an undue expenditure of power or resources. Also, due to the significant volume of data generated by IoT devices in real-time, rapid analysis is necessary to detect security threats. Algorithms like REDNN can perform real-time data analysis without consuming excessive computational power.

Table 6 presents empirical findings comparing the performance of the REDNN model against state-of-the-art techniques utilizing the PT-737E dataset. The results indicate the computational resources required by each model to process each record in the dataset. The REDNN model outperforms other methods by achieving better memory and time resource savings. Specifically, during training, the REDNN model saves more than 99.99% and 99.80% of execution time and memory footprint compared to the SVM model. This is due to the fact that SVM is known to be a computationally expensive ML algorithm, particularly when dealing with large datasets (Catak and Balaban, 2012). Thus, SVM requires more resources than Adaboost and GB decision tree models. As expected, DNN models such as FCNN and REDNN outperform traditional ML models, and this is confirmed by our findings. The results suggest that optimizing DNN models can create an efficient approach with more resource savings than conventional ML methods. This is particularly valuable for building models in an environment with a multi-dimensional and extensive training dataset that requires significant resource savings.

5.2. REDNN model robustness

5.2.1. Robustness against number of epoch

Table 7 illustrates the impact of epoch variation on model robustness against the SH XCS-1003 dataset. The robustness measure is calculated by subtracting the adversarial test accuracy from the clean test accuracy. Our findings indicate that the REDNN model is more robust against each adversarial attack at ten epochs. Specifically, the adversarial accuracy loss of the baseline FCNN is 28.08%, while that of the REDNN is 20.30% against PGD attacks. Although the resilience of both models improves with each epoch increment, the REDNN model exhibits slightly better robustness than the FCNN model during each epoch iteration. This is particularly valuable, as the optimized model can save more resources while thwarting adversarial attacks with both lower and higher epoch iterations. Our results demonstrate that the REDNN model is marginally more robust against adversarial samples than its FCNN counterparts. Therefore, it may be a better option for IoT security due to its enhanced robustness.

Table 7
Effect of number of epoch against models performance with SH XCS-1003 dataset.

Epoch	Model	Clean acc (%)	FGSM acc (%)	PGD acc (%)	Noise acc (%)
10	FCNN	97.73	79.51	69.65	89.52
	REDNN	97.73	86.70	77.43	89.79
20	FCNN	97.73	86.35	77.07	93.86
	REDNN	97.73	86.70	77.43	94.08
40	FCNN	97.73	93.74	86.66	97.08
	REDNN	97.73	94.19	87.10	97.17
60	FCNN	97.73	96.48	90.72	97.63
	REDNN	97.73	96.84	92.09	97.69
80	FCNN	97.73	97.48	94.82	97.72
	REDNN	97.73	97.53	95.34	97.73
100	FCNN	97.73	97.69	97.24	97.73
	REDNN	97.73	97.70	97.29	97.73

5.2.2. Robustness with clipped perturbation samples

Table 8 presents a comparison of models' performance with clipped and non-clipped adversarial samples against randomly chosen datasets. Our findings indicate that in all cases, the performance of detecting FGSM and random noise attacks is better with the clipped procedure compared to the non-clip setting. REDNN outperforms its baseline benchmark in detecting PGD and FGSM, particularly with the Kitsune dataset. The adversarial accuracy losses for both REDNN and FCNN in thwarting non-clipped FGSM adversarial samples of XCS-1003 device data were 0.41% and 0.45%, respectively, with REDNN showing slight improvement. With the same procedure to detect random noise attacks against the Kitsune data, the adversarial accuracy losses of FCNN and REDNN were 4.86% and 0.93%, respectively. These results highlight the robustness of REDNN with clipped and non-clipped adversarial samples, particularly with the Kitsune dataset. Based on these findings, we can suggest REDNN as a model capable of crafting adversarial attacks that are generated using various techniques.

5.2.3. Robustness against model variation

Table 9 presents the performance of REDNN and FCNN using three different hidden layer models architectures. Our results indicate that, across each tested dataset, REDNN resists adversarial attacks better than its baseline. For example, when tested against the Danmini Doorbell dataset, the adversarial accuracy losses of FCNN and REDNN with PGD attacks are 9.18% and 7.23%, respectively. With the optimized four hidden layer model architecture, the adversarial accuracy losses are 1.12% and 0.54% for the FCNN and REDNN models, respectively. These results demonstrate that neural network models with four hidden layers can better detect adversarial attacks. Conversely, models with fewer hidden layers may not stand robust against adversarial attacks. Our findings suggest that REDNN can detect adversarial perturbations regardless of the hidden layers utilized in building the network architecture. As such, REDNN has the potential to be advantageous in an IoT network environment that can be dynamic in terms of architectural settings and security mechanism requirements.

Fig. 3(a) and 3(b) depict the impact of reducing the second hidden layer neuron of each model by 50% and 25% against resilience using the Kitsune dataset. In each setting, REDNN provides better detection

Table 8
Effect of clipping samples against perturbations method.

Dataset	Procedure	Model	Clean acc (%)	FGSM acc (%)	PGD acc (%)	Noise acc (%)
SH XCS-1003-WHT	Clipped	FCNN	97.73	97.69	97.24	97.73
		REDNN	97.73	97.70	97.29	97.73
	Non-clipped	FCNN	97.73	97.24	97.24	97.56
		REDNN	97.73	97.29	97.29	97.58
Danmini Doorbell	Clipped	FCNN	95.11	95.05	93.99	95.11
		REDNN	95.11	95.10	94.57	95.10
	Non-clipped	FCNN	95.11	93.99	93.99	94.79
		REDNN	95.11	94.57	94.57	94.98
Kitsune	Clipped	FCNN	84.09	78.27	70.45	80.67
		REDNN	84.09	83.52	80.18	83.84
	Non-clipped	FCNN	84.09	70.45	70.45	75.81
		REDNN	84.09	80.18	80.18	82.91

Table 9
Variational models perturbations evaluations across datasets.

Dataset	Model	Clean acc (%)	FGSM acc (%)	PGD acc (%)	Noise acc (%)
Danmini Doorbell	FCNN	95.11	91.43	85.93	93.78
	REDNN	95.11	92.93	87.88	94.45
Provision PT-737E	FCNN	92.52	90.31	86.31	91.61
	REDNN	92.52	90.81	87.20	91.91
SH XCS-1002-WHT	FCNN	94.65	92.48	87.87	93.54
	REDNN	94.65	93.21	89.02	93.99
SH XCS-1003-WHT	FCNN	97.73	96.51	92.20	96.98
	REDNN	97.73	96.62	92.33	97.03
Kitsune	FCNN	84.09	75.73	70.02	81.72
	REDNN	84.09	81.56	77.65	83.88

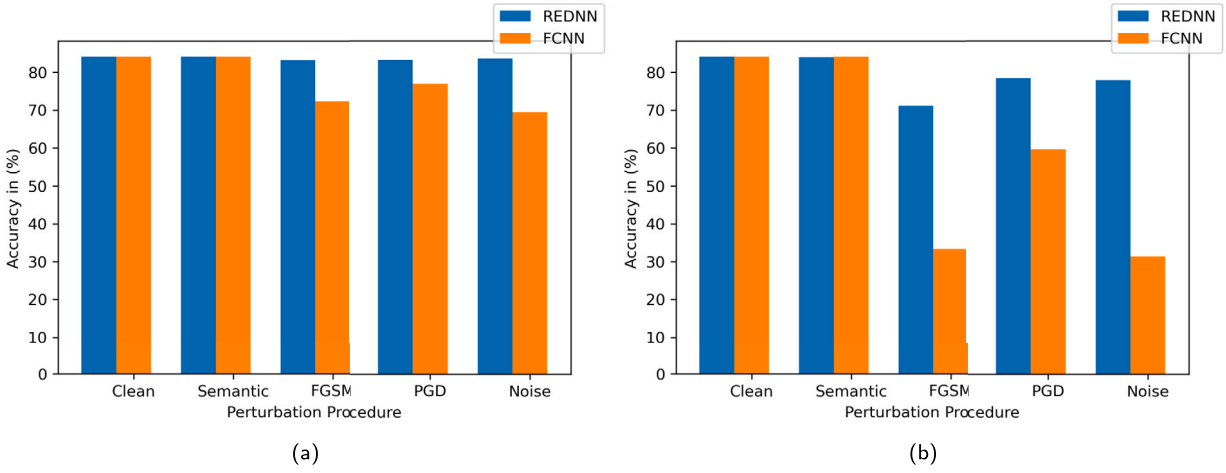


Fig. 3. REDNN vs FCNN accuracy changes with reduce hidden neurons by (a) 50% and (b) 25% against the Kitsune dataset.

accuracy against adversarial samples. As depicted in Fig. 3(a), reducing hidden neuron values affects accuracy, reducing FCNN and REDNN accuracy by 14.66% and 0.42%, respectively. For detecting PGD attacks using the 25% reduced neurons shown in Fig. 3(b), FCNN and REDNN accuracy is reduced by 24.52% and 5.26%, respectively. These results suggest that a significant reduction in hidden neurons affects model resilience to adversarial samples. In each scenario, REDNN is more robust to topology variation than its baseline benchmark. As a result, proper architecture selection can influence the efficient and effective identification of adversarial samples.

Label flipping attacks can be detrimental to the performance of a ML model as they can result in misclassification of data points. Fig. 4 shows the impact of a label flipping attack on the accuracy of the FCNN and REDNN models. Both models were tested against the Kitsune and PT-737E datasets with varying levels of label flipping rates.

The results show that both models can detect and resist label flipping attacks up to a certain rate. In the case of the Kitsune dataset, both models can resist up to a 30% flipping rate, with REDNN outperforming FCNN at a 40% rate. On the other hand, for the PT-737E dataset, FCNN's accuracy reduces significantly at a 50% flipping rate, while REDNN maintains its performance up to the same rate.

These results demonstrate the robustness of REDNN against label flipping attacks, especially in the PT-737E dataset. The regularization properties of the REDNN model can make it less susceptible to slight changes in the training data, making it more resilient against poisoning attacks.

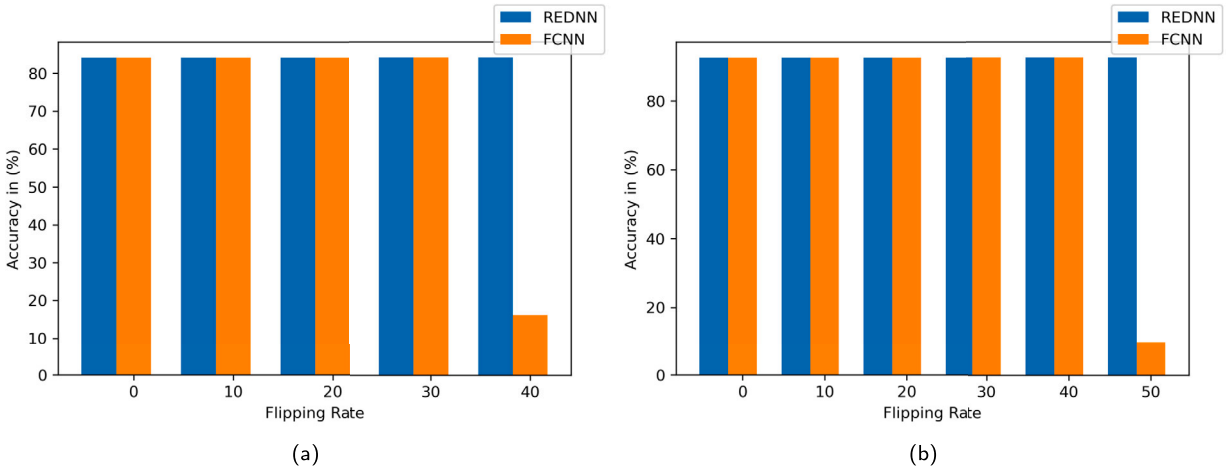


Fig. 4. REDNN vs FCNN accuracy changes with label flip against (a) Kitsune and (b) PT-737E dataset.

Table 10
Performance evaluation comparison with Provision PT-737E dataset.

Model	Clean acc (%)	Noise acc (%)	Poisoned label acc (%)
SVM	92.52	70.89	7.48
GB	92.58	61.91	10.01
Adaboost	92.47	53.31	11.05
FCNN	92.52	91.57	9.55
REDNN	92.52	91.87	92.52

Table 11
Model resilience evaluation with Kitsune dataset.

Attacks	Model	Acc (%)	Precision	Recall	F1 score
FGSM	FCNN	83.60	0.8408	0.9744	0.9027
	REDNN	84.09	0.8409	1.0000	0.9136
PGD	FCNN	82.34	0.8408	0.9744	0.9027
	REDNN	84.09	0.8409	1.0000	0.9136
Noise	FCNN	76.67	0.8412	0.8906	0.8652
	REDNN	83.73	0.8411	0.9944	0.9113

In addition to its significant resource savings capability, REDNN demonstrates greater resilience against random noise attacks when compared to each of the models analyzed as shown in Table 10. Subsequently, we examined the impact of poisoning 50% of the training data through label modification (refer to the Poisoned label column). This resulted in a reduction in the robustness of the SVM, GB, Adaboost, and FCNN models, with adversarial accuracy losses of 85.04%, 82.57%, 81.42%, and 82.97%, respectively. The results reveal that REDNN exhibits better resistance to label poisoning attacks, without any loss in adversarial accuracy. This finding implies that a stable and less complex model may be able to overcome label poisoning attacks, which are a significant threat in a detection system since attackers can easily manipulate the data. Furthermore, these results demonstrate the superior robustness of the REDNN model compared to conventional ML models and answers RQ2. As such, REDNN may be a suitable solution for IoT security monitoring or efficient ML-based security systems.

Table 11 illustrates the performance of the models evaluated in terms of test set accuracy, precision, recall, and harmonic score (F1) while exploring the impact of FP16 integration on model resilience. The implementation of FP16 has a significant impact on the robustness of the FCNN model, particularly in its ability to withstand random noise attacks, resulting in adversarial accuracy and F1-score losses of 7.06% and 4.61%, respectively, when compared to REDNN. As previously

Table 12
Federated model training memory consumption between REFDNN and BFDNN (cumulative).

Dataset	Model	Memory MB	Time mins	Test set acc %
Danmini Doorbell	BFDNN	3.783	0.099	95.11
	REFDNN	0.857	0.081	95.11
Ecobee Thermostat	BFDNN	3.732	0.091	93.36
	REFDNN	0.815	0.071	93.36
Ennio Doorbell	BFDNN	4.147	0.090	88.94
	REFDNN	0.805	0.074	88.94
Provision PT-737E	BFDNN	3.463	0.092	92.52
	REFDNN	0.853	0.077	92.52
Provision PT-838	BFDNN	3.423	0.085	88.07
	REFDNN	0.814	0.074	88.07
Samsung SNH-1011-N	BFDNN	3.783	0.099	86.06
	REFDNN	0.858	0.081	86.06
SimpleHome XCS-1002	BFDNN	3.494	0.090	94.65
	REFDNN	0.816	0.072	94.65
SimpleHome XCS-1003	BFDNN	3.914	0.085	97.73
	REFDNN	0.801	0.071	97.73
Wustl	BFDNN	3.002	0.095	94.26
	REFDNN	0.816	0.076	94.26

mentioned in this paper, ML engineers frequently choose low-precision implementations to reduce computation time and memory usage during model training and testing, but this comes at the cost of sacrificing overall accuracy. The results demonstrate that REDNN exhibits better resilience in countering each adversarial attack. Furthermore, the findings suggest that FP16 implementation has only a minor impact on the robustness of the REDNN model, making it a more effective and resilient IoT security monitoring technique compared to its FCNN counterparts. The results indicate that REDNN possesses attack resilience capabilities, even when integrated with FP16, which can potentially degrade model performance. Overall, resource-efficient ML algorithms like REDNN can effectively address the unique challenges of IoT security monitoring by providing scalable, real-time, and efficient analysis of data derived from multiple IoT devices. IoT security monitoring requires analyzing data from a large number of devices simultaneously. Resource-efficient ML algorithms like REDNN can be easily scaled to handle large volumes of data from multiple devices in federated settings to accomplish this.

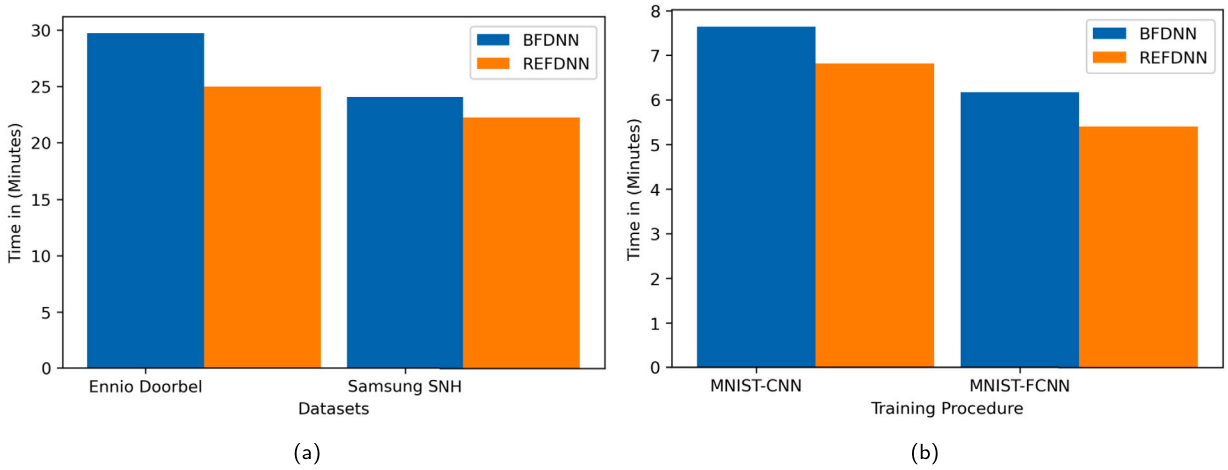


Fig. 5. Federated model training execution time of REFDNN and BFDNN against (a) IoT and (b) MNIST datasets.

Table 13
Simulated federated training performance comparison between BFDNN and REFDNN with MNIST dataset.

Procedure	Model	Time mins	Time save (%)	Test set acc %
FCNN-MNIST	BFDNN	1.393	N/A	34.64
	REFDNN	1.346	3.374	91.03
CNN-MNIST	BFDNN	1.583	N/A	90.59
	REFDNN	1.457	7.960	98.28

Table 14
Federated model accuracy: REFDNN vs BFDNN against CNN-MNIST training procedure.

Federated rounds	Model	Test set acc (%)
50 - 1 epoch	REFDNN	97.00
	BFDNN	89.00
50 - 2 epoch	REFDNN	99.00
	BFDNN	93.00
100 - 1 epoch	REFDNN	97.00
	BFDNN	89.00
100 - 2 epoch	REFDNN	99.00
	BFDNN	93.00

5.3. REFDNN model training performance (decentralized manner)

5.3.1. Simulated workers FL scenario

Table 12 shows the memory footprint and time usage for each dataset in a FL setting. REFDNN exhibits lower runtime and memory footprints across all datasets. Notably, the accuracy of both REFDNN and BFDNN remained constant across each benchmark dataset. In terms of client processing runtime, REFDNN is more efficient, indicating less complexity, faster learning capability, and superior resource savings performance compared to BFDNN. Due to these resource savings, REFDNN may be a better option for IoT security monitoring, especially for on-device learning on a diverse range of resource-constrained edge devices.

As a generic solution for on-device learning, it is important to assess the method's performance on non-IoT datasets (MNIST) (see Table 13). This can also allow us to leverage REFDNN's resource-saving capability with CNN, which provides accurate performance in image classification. PySyft WS simulated workers were used to examine the performance of the BFDNN and REFDNN techniques in each federated training. This was done to assess REFDNN's performance using a simulated network with a client and server scenario running on the same machine, unlike PySft virtual workers counterparts that run as constructs within the same python program. As expected, with each DNN (CNN and FCNN) variant, REFDNN demonstrates better accuracy than its BFDNN counterparts. The better performance on the MNIST dataset is due to the regularization and optimization of REFDNN. Furthermore, it produces lower training execution time. These results demonstrate the importance of regularization (Krueger and Memisevic, 2015) and (Lever et al., 2016) on accuracy against DNN variants and warrant further investigation in realistic settings.

5.3.2. Network workers FL testbed results

Fig. 5(a) shows that REFDNN has a faster estimated convergence time than BFDNN when training on the Ennio Doorbell and Samsung SNH IoT datasets on the GB-BXBT-2807 testbed. This indicates that REFDNN is more efficient in detecting IoT attacks in real-time, which is beneficial in resource-constrained environments. Similarly, Fig. 5(b) shows that REFDNN is more computationally efficient than BFDNN when training on the MNIST dataset, with the FCNN variant of REFDNN being particularly appropriate for on-device learning in resource-constrained IoT environments. These results suggest that REFDNN is a more suitable method for deployment in IoT resource environments, where resource savings are a priority.

Table 14 presents a performance comparison between REFDNN and BFDNN with the federated training procedure CNN-MNIST over 100 and 50 communication rounds. The reported results pertain to the use of one and two local epoch iterations. Across each epoch of every communication round, REFDNN exhibited superior accuracy compared to its baseline counterparts. These outcomes imply that REFDNN is proficient in the classification of both IoT and non-IoT datasets in real-time, exhibiting greater accuracy than its alternatives.

6. Conclusion

This research introduces REDNN, a deep neural network-based approach specifically designed to detect cyberattacks on IoT devices while prioritizing resource efficiency. The effectiveness of this approach is evaluated through experimentation using eleven benchmark datasets. The results demonstrate that REDNN exhibits robustness against adversarial attacks, accurately detects cyberattacks on IoT networks, and significantly conserves resources. Furthermore, this study presents a resource-efficient federated learning model called REFDNN, tailored for IoT security monitoring. The effectiveness of REFDNN is assessed using eight IoT datasets and one MNIST image dataset, both in virtual and real-world testbed setups. Future research endeavors will focus on investigating the detection capabilities of the proposed algorithms against real-time attacks and evaluating the resilience of REFDNN in practical IoT and cyber-physical network environments that involve a large number of edge devices.

CRediT authorship contribution statement

Idris Zakariyya: Conceptualization, Methodology, Software. **Harsha Kalutarage:** Supervision (principal supervisor). **M. Omar Al-Kadri:** Supervision (second supervisor).

Declaration of competing interest

Idris Zakariyya reports financial support was provided by Petroleum Technology Development Fund.

Data availability

No data was used for the research described in the article.

Acknowledgement

This work was supported by the Petroleum Technology Development Fund (PTDF), Nigeria.

References

Abiodun, O.I., Jantan, A., Omolara, A.E., Dada, K.V., Mohamed, N.A., Arshad, H., 2018. State-of-the-art in artificial neural network applications: a survey. Heliyon 4, e00938.
Abou Khamis, R., Matrawy, A., 2020. Evaluation of adversarial training on different types of neural networks in deep learning-based IDSs. In: 2020 International Symposium on Networks, Computers and Communications. ISNCC. IEEE, pp. 1–6.
Aggarwal, C.C., et al., 2018. Neural Networks and Deep Learning, vol. 10. Springer. 978-3.
Aloraini, F., Javed, A., Rana, O., Burnap, P., 2022. Adversarial machine learning in IoT from an insider point of view. J. Inf. Secur. Appl. 70, 103341.
Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., et al., 2017. Understanding the Mirai botnet. In: 26th {USENIX} Security Symposium. {USENIX } Security 17, pp. 1093–1110.
Athalye, A., Carlini, N., Wagner, D., 2018. Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples. In: International Conference on Machine Learning. PMLR, pp. 274–283.
Baldominos, A., Saez, Y., Isasi, P., 2019. A survey of handwritten character recognition with MNIST and EMNIST. Appl. Sci. 9, 3169.
Bojan, J., 2022. Internet of things statistics for 2022 – taking things apart. https://dataprot.net/statistics/iot-statistics/.
Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, H.B., et al., 2019. Towards federated learning at scale: system design. arXiv preprint. arXiv:1902.01046.
Bosman, A., Engelbrecht, A., Helbig, M., 2018. Fitness landscape analysis of weight-elimination neural networks. Neural Process. Lett. 48, 353–373.
Catak, F.O., Balaban, M.E., 2012. CloudSVM: training an SVM classifier in cloud computing systems. In: Joint International Conference on Pervasive Computing and the Networked World. Springer, pp. 57–68.
David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Wang, T., et al., 2021. Tensorflow lite micro: embedded machine learning for tinyML systems. Proc. Mach. Learn. Syst. 3, 800–811.
Deng, L., 2012. The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Process. Mag. 29, 141–142.
Elrawy, M.F., Awad, A.I., Hamed, H.F., 2018. Intrusion detection systems for IoT-based smart environments: a survey. J. Cloud Comput. 7, 1–20.
Han, S., Pool, J., Tran, J., Dally, W., 2015. Learning both weights and connections for efficient neural network. Adv. Neural Inf. Process. Syst. 28.
He, C., Mushtaq, E., Ding, J., Avestimehr, S., 2022. FedNAS: federated deep learning via neural architecture search. https://openreview.net/forum?id=1OHZX4YDqhT.
Hosseini, H., Xiao, B., Jaiswal, M., Poovendran, R., 2017. On the limitation of convolutional neural networks in recognizing negative images. In: 2017 16th IEEE International Conference on Machine Learning and Applications. ICMLA. IEEE, pp. 352–358.
Hsu, H.T., Jong, G.J., Chen, J.H., Jhe, C.G., 2019. Improve IoT security system of smart-home by using support vector machine. In: 2019 IEEE 4th International Conference on Computer and Communication Systems. ICCCS. IEEE, pp. 674–677.
Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q.V., Wu, Y., et al., 2019. GPipe: efficient training of giant neural networks using pipeline parallelism. Adv. Neural Inf. Process. Syst. 32, 103–112.
Iandola, F., Keutzer, K., 2017. Keynote: small neural nets are beautiful: enabling embedded systems with small deep-neural-network architectures. In: 2017 International Conference on Hardware/Software Codesign and System Synthesis. CODES+ ISSS. IEEE, pp. 1–10.
Ide, H., Kurita, T., 2017. Improvement of learning for CNN with ReLU activation by sparse regularization. In: 2017 International Joint Conference on Neural Networks. IJCNN. IEEE, pp. 2684–2691.
Imteaj, A., Thakker, U., Wang, S., Li, J., Amini, M.H., 2021. A survey on federated learning for resource-constrained IoT devices. IEEE Int. Things J.
intel, 2020. Choose precision. https://software.intel.com/content/www/us/en/develop/articles/should-i-choose-fp16-or-fp32-for-my-deep-learning-model.html.
Jenalea, H., 2017. Number of connected IoT devices will surge to 125 billion by 2030, IHS markit says. https://news.ihsmarkit.com/prviewer/release_only/slug/number-connected-iot-devices-will-surge-125-billion-2030-ihs-markit-says.
Jiang, Y., Wang, S., Valls, V., Ko, B.J., Lee, W.H., Leung, K.K., Tassiulas, L., 2019. Model pruning enables efficient federated learning on edge devices. arXiv preprint. arXiv:1909.12326.
Johansson, R., 2018. Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib. Apress.
Kodali, S., Hansen, P., Mulholland, N., Whatmough, P., Brooks, D., Wei, G.Y., 2017. Applications of deep neural networks for ultra low power IoT. In: 2017 IEEE International Conference on Computer Design. ICCD. IEEE, pp. 589–592.
Komer, B., Bergstra, J., Eliasmith, C., 2019. Hyperopt-sklearn. In: Automated Machine Learning. Springer, Cham, pp. 97–111.
Krueger, D., Memisevic, R., 2015. Regularizing RNNs by stabilizing activations. arXiv preprint. arXiv:1511.08400.
Kshetri, N., 2021. Economics of artificial intelligence in cybersecurity. IT Prof. 23, 73–77.
Kurakin, A., Goodfellow, I., Bengio, S., 2016. Adversarial machine learning at scale. arXiv preprint. arXiv:1611.01236.
Lever, J., Krzywinski, M., Altman, N., 2016. Points of significance: regularization. Nat. Methods 13, 803–805.
Li, X., Liu, H., Wang, W., Zheng, Y., Lv, H., Lv, Z., 2022. Big data analysis of the internet of things in the digital twins of smart city based on deep learning. Future Gener. Comput. Syst. 128, 167–177.
Lim, W.Y.B., Luong, N.C., Hoang, D.T., Jiao, Y., Liang, Y.C., Yang, Q., Niyato, D., Miao, C., 2020. Federated learning in mobile edge networks: a comprehensive survey. IEEE Commun. Surv. Tutor. 22, 2031–2063.
Liu, Y., Kumar, N., Xiong, Z., Lim, W.Y.B., Kang, J., Niyato, D., 2020. Communication-efficient federated learning for anomaly detection in industrial internet of things. In: GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE, pp. 1–6.
Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., 2020. IoT type-of-traffic forecasting method based on gradient boosting neural networks. Future Gener. Comput. Syst. 105, 331–345.
Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., Elovici, Y., 2018. N-BaioT—network-based detection of IoT botnet attacks using deep autoencoders. IEEE Pervasive Comput. 17, 12–22.
Merenda, M., Porcaro, C., Iero, D., 2020. Edge machine learning for AI-enabled IoT devices: a review. Sensors 20, 2533.
Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A., 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. arXiv preprint. arXiv:1802.09089.
Nguyen, T.D., Marchal, S., Miettinen, M., Fereidooni, H., Asokan, N., Sadeghi, A.R., 2019. Dïot: a federated self-learning anomaly detection system for IoT. In: 2019 IEEE 39th International Conference on Distributed Computing Systems. ICDCS. IEEE, pp. 756–767.
Oyama, Y., Ben-Nun, T., Hoefler, T., Matsuoka, S., 2018. Accelerating deep learning frameworks with micro-batches. In: 2018 IEEE International Conference on Cluster Computing. CLUSTER. IEEE, pp. 402–412.
Papernot, N., Faghri, F., Carlini, N., Goodfellow, I., Feinman, R., Kurakin, A., Xie, C., Sharma, Y., Brown, T., Roy, A., et al., 2020. Technical report on the cleverhans v2. 1.0 adversarial examples library. arXiv 2018. arXiv preprint. arXiv:1610.00768.
Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., 2019. Pytorch: an imperative style, high-performance deep learning library. Adv. Neural Inf. Process. Syst. 32, 8026–8037.
Pedregosa, F., Gervais, P., 2019. Memory profiler (Python). Python Software Foundation. https://pypi.org/project/memory-profiler/ (Accessed March 25).
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al., 2011. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.
Pitropakis, N., Panaousis, E., Giannetsos, T., Anastasiadis, E., Loukas, G., 2019. A taxonomy and survey of attacks against machine learning. Comput. Sci. Rev. 34, 100199.
Popoola, S.I., Ande, R., Adebisi, B., Gui, G., Hammoudeh, M., Jogunola, O., 2021. Federated deep learning for zero-day botnet attack detection in IoT edge devices. IEEE Int. Things J.
Preuveneers, D., Rimmer, V., Tsingenopoulos, I., Spooren, J., Joosen, W., Ilie-Zudor, E., 2018. Chained anomaly detection models for federated learning: an intrusion detection case study. Appl. Sci. 8, 2663.
Pujari, M., Pacheco, Y., Cherukuri, B., Sun, W., 2022. A comparative study on the impact of adversarial machine learning attacks on contemporary intrusion detection datasets. SN Comput. Sci. 3, 1–12.
Pumperla, M., 2018. Hyperas: Hyperopt: a very simple and convenient wrapper for hyperparameter optimization. https://github.com/maxpumperla/hyperas.
Ryffel, T., Trask, A., Dahl, M., Wagner, B., Mancuso, J., Rueckert, D., Passerat-Palmbach, J., 2018. A generic framework for privacy preserving deep learning. arXiv preprint. arXiv:1811.04017.
Sánchez, P.M.S., Valero, J.M.J., Celdrán, A.H., Bovet, G., Pérez, M.G., Pérez, G.M., 2021. A survey on device behavior fingerprinting: data sources, techniques, application scenarios, and datasets. IEEE Commun. Surv. Tutor. 23, 1048–1077. https://doi.org/10.1109/COMST.2021.3064259.

Shafahi, A., Huang, W.R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., Goldstein, T., 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. Adv. Neural Inf. Process. Syst. 31.

Shen, S., Li, R., Zhao, Z., Liu, Q., Liang, J., Zhang, H., 2020. Efficient deep structure learning for resource-limited IoT devices. In: GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE, pp. 1–6.

Tang, D., Tang, L., Dai, R., Chen, J., Li, X., Rodrigues, J.J., 2020. MF-Adaboost: LDoS attack detection based on multi-features and improved Adaboost. Future Gener. Comput. Syst. 106, 347–359.

Tang, J., Sun, D., Liu, S., Gaudiot, J.L., 2017. Enabling deep learning on IoT devices. Computer 50, 92–96.

Teixeira, M.A., Salman, T., Zolanvari, M., Jain, R., Meskin, N., Samaka, M., 2018. SCADA system testbed for cybersecurity research using machine learning approach. Future Internet 10, 76.

TensorFlow, 2022. float16 quantization. https://www.tensorflow.org/lite/performance/post_training_float16_quant.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., McDaniel, P., 2017. Ensemble adversarial training: attacks and defenses. arXiv preprint. arXiv:1705.07204.

Vinayakumar, R., Barathi Ganesh, H.B., Prabaharan, P., et al., 2018. Deep-net: deep neural network for cyber security use cases. arXiv preprint. arXiv:1812.03519.

Yang, Q., Liu, Y., Cheng, Y., Kang, Y., Chen, T., Yu, H., 2019. Federated learning. Synth. Lect. Artif. Intell. Mach. Learn. 13, 1–207.

Zakariyya, I., 2021. Resource efficient IoT DNNs algorithm. https://github.com/izakariyya/R_DNN_IoT.

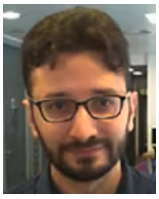
Zakariyya, I., 2022. Resource efficient federated algorithm with realistic workers. https://github.com/izakariyya/testbd-fl-iot.

Zandberg, K., Schleiser, K., Acosta, F., Tschofenig, H., Baccelli, E., 2019. Secure firmware updates for constrained IoT devices using open standards: a reality check. IEEE Access 7, 71907–71920.

Zhang, Y., Krishnan, V., Pi, J., Kaur, K., Srivastava, A., Hahn, A., Suresh, S., 2019. Cyber physical security analytics for transactive energy systems. IEEE Trans. Smart Grid 11, 931–941.

Idris Zakariyya is a highly accomplished individual in the field of Computer Science (Cybersecurity), holding a B.Sc. degree in Computer Science from Bayero University, Kano, Nigeria (2011), and an M.Sc. degree in Computer Science from Sultan Zainal Abidin University, Terengganu, Malaysia (2015). In 2022, he successfully obtained his Ph.D. in Cybersecurity from Robert Gordon University in the United Kingdom and has authored nine technical papers in prominent publications. Idris's research interests focus on Internet of Thinsg (IoT) Cybersecurity monitoring, AI-based IoT network security monitoring, Intrusion detection, On-device learning, and Adversarial Robustness of AI, thus contributing significantly to the advancement of Cybersecurity.

Harsha Kumara Kalutarage is a lecturer in Cyber Security in the School of Computing at Robert Gordon University in the UK. He has 10+ years of research experience in Cybersecurity and has produced 40+ publications in this area. His research interests span AI & Security, the use of AI for security applications and studying the security of AI-enabled systems. Harsha holds a Ph.D. in Computing (Cyber Security), an M.Phil. in Computer Science (NLP) and a B.Sc. Special (Hons) degree in Statistics & Computer Science.



M. Omar Al-Kadri received his B.Eng. in Computer Engineering from IUST, Syria, in 2010, M.Sc. (with distinction) in Networking and Data communication from Kingston University, UK in 2013, and Ph.D in Telecommunication engineering from Kings College London, UK, in 2017. He is now a senior lecturer in networking and Cybersecurity at Birmingham City University, UK. His current research interests include security of wireless communications with application to healthcare, security of vehicular networks, full-duplex communications, HetNets, and MAC/routing protocols.