Resource efficient federated deep learning for IoT security monitoring.

ZAKARIYYA, I., KALUTARAGE, H. and AL-KADRI, M.O.

2022

Resource Efficient Federated Deep Learning for IoT Security Monitoring

Idris Zakariyya[1 [0000−0002−7983−1848]], Harsha Kalutarage[1 [0000−0001−6430−9558]], and M. Omar Al-Kadri[2 [0000−0002−1146−1860]]

[1] School of Computing, Robert Gordon University, UK
{i.zakariyya, h.kalutarage}@rgu.ac.uk
[2] School of Computing and Digital Technology, Birmingham City University, UK
omar.alkadri@bcu.ac.uk

Abstract. Federated Learning (FL) uses a distributed Machine Learning (ML) concept to build a global model using multiple local models trained on distributed edge devices. A disadvantage of the FL paradigm is the requirement of many communication rounds before model convergence. As a result, there is a challenge for running on-device FL with resource-hungry algorithms such as Deep Neural Network (DNN), especially in the resource-constrained Internet of Things (IoT) environments for security monitoring. To address this issue, this paper proposes Resource Efficient Federated Deep Learning (REFDL) method. Our method exploits and optimizes Federated Averaging (Fed-Avg) DNN based technique to reduce computational resources consumption for IoT security monitoring. It utilizes pruning and simulated micro-batching in optimizing the Fed-Avg DNN for effective and efficient IoT attacks detection at distributed edge nodes. The performance was evaluated using various realistic IoT and non-IoT benchmark datasets on virtual and testbed environments build with GB-BXBT-2807 edge-computing-like devices. The experimental results show that the proposed method can reduce memory usage by 81% in the simulated environment of virtual workers compared to its benchmark counterpart. In the realistic testbed scenario, it saves 6% memory while reducing execution time by 15% without degrading accuracy.

Keywords: Distributed machine learning · Edge devices · Federated learning (FL) · Deep Neural Network (DNN) · Internet of Things (IoT), Security Monitoring.

1 Introduction

The Internet of Things (IoT) is an ecosystem that consists of multiple intelligent devices. The Markit estimates suggest that 125 billion devices will be part of IoT by 2030 [1]. The connected IoT devices are potentially used in smart-home, smart cities, intelligent automation and cyber-physical systems. These devices used embedded systems, such as processors, sensors and communication hardware to collect and exchange data. These devices share the collected data with other

connected edge devices. The shared data can improve data management and monitoring, human-machine interaction, and Artificial Intelligence (AI) analytics. However, these devices are becoming potential avenues for various cyber attacks and other cyber mafias. For instance, the massive Distributed Denial-of-Service (DDoS) attack on insecure IoT devices powered by a virus called Mirai (Linux. Gafgyt) causes a severe disaster [2]. At the same time, these IoT devices consist of low computational power, and limited memory and processors. Because of that, AI techniques developed for mainstream and other general purposes computing devices cannot be deployed on resource-constrained IoT devices. Therefore, the mechanisms to address security challenges in the IoT and cyber-physical systems need to be resource-efficient and effective, especially in Federated Learning (FL) scenario that augments data security and privacy issues.

Recent research has shown the potential applications of ML algorithms, especially Deep Neural Network (DNN), in cyber security monitoring [3]. However, IoT devices are resource-constrained and distributed in nature hence DNN-based cyber security scheme cannot be directly deployed for security monitoring in IoT environments. In addition, organizations are concerned about privacy in data sharing for training AI-based techniques in a centralised manner (e.g. data centre). In this aspect, FL [4] approach provides a promise but may not scale through IoT and cyber-physical devices because client edge devices are usually more resource-constrained in terms of storage, computational power, communication bandwidth and memory than server machines in the data centre. Therefore, training a federated DNN model consisting of millions of parameters on resource-constrained IoT devices is a challenge. To this end, we investigate the following research questions (RQs) to develop a suitable federated DNN-based method for the security monitoring of resource-constrained environments such as IoT.

RQ1: Can existing DNNs be trained efficiently in FL settings so that the resulting model can be appropriate for IoT security monitoring in resource-constrained environments? (see section 3.2)

RQ2: Can the resulting Resource Efficient Federated DNN (REFDL) effectively and accurately detect attacks on IoT networks without accuracy degradation? (see section 5)

For our experiments, we utilize a Federated Averaging (FedAvg) DNN along with eight IoT benchmark datasets to build an REFDL model. The experimental results are encouraging as the resulting REFDL shows lower memory consumption with better classification performance in simulated and real testbed federated settings against each data set used in our experiments. The federated integration of the model also helps to preserve the privacy of IoT device data during on-device model training.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3.2 describes the proposed method and the utilized FL technique, while Section 4 describes the evaluation process. Results and discussion can be found in Section 5. Finally, Section 6 concludes the paper with future research directions.

2 Related Work

This section presents related studies concerning deep learning for IoT security monitoring, followed by brief descriptions of FL and its applications to IoT environments.

Deep Neural Network in IoT. Significant research has been conducted on IoT security monitoring using AI techniques. Most of these methods utilized DNN [5]. Mohammad et al. [6] used DNN for IoT data analysis and network traffic classification tasks. Li et al. [7] carried out similar tasks using IoT smart cities data. Shen et al. [8] proposed compact structure-based learning with Convolutional Neural Network (CNN) for an IoT resource-constrained environment. The technique demonstrates its potentiality on the CIFAR-10 and Imagenet benchmark datasets. The lack of model assessments with IoT benchmark datasets and non-consideration of memory usage are the restrictions of their method's potentiality for deployment in a resource-constrained environment. Rock et al. [9] quantized CNN for inference on radar sensor data. Kodali et al. [10] exploit the potentiality of DNN, specifically Fully Connected Neural Network (FCNN), for classification tasks on ultra low power IoT devices. The aim is to improve the detection performance without reducing the model complexity. The lack of consideration for model complexity while selecting the FCNN architecture may restrict method feasibility. In addition, most of the stated optimization approaches considered the quantization of weights and bias parameters. Zakariyya et al. [11] proposed resource-efficient and robust DNN methods for IoT security monitoring in centralized settings. But, our proposed approach in this paper aims to reduce memory and time usage without degrading accuracy significantly in decentralized settings. The method exploits pruning, simulated micro-batching and parameter regularization to optimize the resulting model in terms of memory requirements and accuracy performance. This is useful, especially for the task of distributed learning in a resource-constrained environment.

FL in IoT Environments. McMahan et al. [12] proposed the first FedAvg FL technique that enables the training of a local model on multiple clients without sharing the client's local data to a server. This technique offers a promise in terms of model convergence with various client local data in non-independent and non-identically distributed settings. For this reason, researchers from several disciplines explored FL methods from different perspectives. In the field of IoT security monitoring, FL is gaining popularity. Preuveneers et al. [13] described FL applications for intrusion detection in IoT networks. Imteaj et al. [14] described the open research directions regarding the FL applications on resource-constrained IoT devices. Thein et al. [15] described the capability of FL in detecting attacks on industrial IoT devices. Liu et al. [16] enhance that investigation by considering raw sensor reading data. Jiang et al. [17] utilized model pruning for efficient FL training on edge devices. Bonawitz et al. [18] proposed a scalable FL framework for mobile devices to reduce communication overhead. Popoola et al. [19] used FL to detect a zero-day attack in an IoT network environment. Their implementation take the advantage of FL data privacy without considering resource limitations. Zakariyya et al. [20] used model parameters pruning and data parallelism (micro-

batching) in optimizing FL to reduce memory consumption on IoT networks. However, none of the mentioned proposals considers optimizing FL to save memory and time resources using different DNN variants. We address this challenge by optimizing the federated training procedure using raw network traffic datasets from various IoT devices. Then, we proposed a REFDL method with minimal resource consumption. This method maintains state-of-the-art accuracy while reducing memory and time consumption in both simulated and real embedded devices experimental settings.

3 Methodology

To demonstrate the proof of concept, we will use Baseline Federated Deep Learning (BFDL) with FCNN and CNN model variation against some IoT and non-IoT benchmark datasets and exploit the BFDL optimization algorithm to obtain the REFDL. We demonstrate that careful optimization of the BFDL algorithm is sufficient to produce the REFDL. The efficient REFDL can detect attack activities on IoT and accurate classification with non-IoT datasets.

3.1 Baseline Federated Deep Learning (BFDL)

The BFDL utilized the classical FedAvg algorithm with an integrated DNN (FCNN / CNN) model. The in-cooperated DNN is a neural network containing deep layers of neurons representing the input data. These neurons correspond to the computing units that can transmit computational results operated with their activation function and the input. The FCNN is a sequential form of DNN that connects neurons with the corresponding weights and bias parameters. The weights and biases serve as information storage components. The baseline model of the BFDL ($\mathcal{M}_n$) in Algorithm 1 consists of network topology, activation functions and corresponding values for weights and bias. The weight and bias values settings can minimize the error function $\mathcal{E}_{\mathcal{M}_n}$ evaluated over the labelled training data $\mathcal{D}_{tr}$. This procedure can built a single master model of the FedAvg algorithm that can serve as the aggregated of the client models. The function BASE in line 1 of Algorithm 1 describes the $\mathcal{M}_n$ training using a Stochastic Gradient Descent (SGD) algorithm with backpropagation [21] in FL scenario. At each communication round, the server in function Device UPDATE of Algorithm 1 is capable of distributing a master model to each client's subsets. Each client performs iterative rounds of gradient descent weights update with their local data and returns to the server in Algorithm 3. This is determined to minimize the cost function in Equation 1 and Equation 2 in-order to create a global master model. Then, the execution time and memory footprints are estimated based on lines 11 and 12 of Algorithm 1. These are the records of training resource usage at the device level after the local weights update. As expressed in line 17, computed model weights are returned to the coordinating server in Algorithm 3. The server is responsible for averaging the return weight for global model aggregation. With a function that learns from $\mathcal{D}_{tr}$, the global model can appropriately map unseen

samples. The resulting BFDL approach uses supervised DNN (FCNN and CNN) as a classifier, $\mathcal{M}_n$ can accept an input $\mathcal{D}_{tr}$ and outputs a probability class of vector $\hat{Y}$. The desired output $\hat{Y}$ are rounded up to the closest integer using a specified threshold value t as in Equation 3. This output represents either the benign (1) or the attack (0) traffic instance against the IoT data or representative class for the image dataset.

Algorithm 1 Baseline BFDL training

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

function BASE($\mathcal{D}_{tr}[\,]$) ▷ Training baseline model
  for $i = 1$ to $\mathcal{T}$ **do**
    Mini-batch $B = \{(x_1, y_1), ..., (x_m, y_m\ )\} \subset D_{tr}$
    $F_p(B)$ ▷ Forward propagation with B
    $\mathcal{E}_i \leftarrow L$ ▷ L = Base loss
    B_p(B) ▷ Backward propagation
    function DEVICE UPDATE((d)) ▷ Run on device d
      $B_s \leftarrow$ (data P_d in batches of size B)
      for batch $b \in B_s$ **do**
        $w \leftarrow$ local weights update ▷ device local weights update computation
        Estimate m_i ▷ Execution memory at epoch i
        Estimate t_i ▷ Execution time at epoch i
        $\mathcal{M}_n$ = Trained model that estimate $\mathcal{E}_i, m_i, t_i$
      end for
    end function
  end for
  return w to server in Alg. 3 ▷ Calls to coordinating server in Alg. 3 for weights averaging
  return $(\mathcal{M}_n, \mathcal{E}_i, m_i, t_i)$
end function

$$J(W, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{Y}^i, Y^i) \tag{1}$$

$$L(\hat{Y}^i, Y^i) = -(Y \log \hat{Y} + (1 - Y) \log (1 - \hat{Y})) \tag{2}$$

$$Output = \begin{cases} 0 \text{ if } \hat{Y} \leq t \\ 1 \text{ if } \hat{Y} > t \end{cases} \tag{3}$$

3.2 Resource Efficient Federated Deep Learning (REFDL)

As mentioned above, training a resource-efficient DNN model for FL task can be a challenging task, especially in IoT security monitoring [22]. Because of the FL

communication rounds and DNN model parameters requirements in designing and building the desirable architecture. The complexity of such an approach increases with multidimensional datasets.

Algorithm 2 Proposed method to obtain REFDL

Input: Penalty term λ, ($\mathcal{D}_{tr}, \mathcal{T}, B, L$, in Alg. 1)
Output: Efficient model $\mathcal{M}_e$

1: **function** EFFICIENT($\mathcal{D}_{tr}[\,]$)
2: **for** $j = 1$ to $\mathcal{T}$; **do**
3: Micro-batch $M = \{(x_1, y_1), ..., (x_m, y_m)\} \subset B$
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: $M_s \leftarrow$ (data P_d in batches of size 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: $m_{tr} = m_j$ ▷ m_tr = Efficient memory
19: $t_{tr} = t_j$ ▷ t_{tr} = Efficient time
20: $\mathcal{M}_e$ = Trained model that estimate $\mathcal{E}_j, m_{tr}, t_{tr}$
21: **end if**
22: **end if**
23: **end for**
24: **end function**
25: **end for**
26: **return** w to server in Alg. 3 ▷ Calls to coordinating server in Alg. 3 for weights averaging
27: **return** $(\mathcal{M}_e, \mathcal{E}_j, m_{tr}, t_{tr})$
28: **end function**

To this end, we utilize the baseline model in BFDL to produce its resource-efficient counterparts (REFDL). The training procedure described in Algorithm 2 optimizes a function using $\mathcal{D}_{tr}$ in the FL scenario to obtain the efficient M_e corresponding to the REFDL model. As described in line 3 in Algorithm 2, the optimization procedure utilized micro-batching [23], which is suitable for breaking a large amount of data into smaller batches for efficient on-device model training. Unlike the mini-batch, the micro-batching is particularly suitable for most datasets, especially the IoT ones. To reduce network complexity, we used

a penalty [24] (weight elimination) technique with a threshold parameter w_0 as shown in regularized Equation 4. This is a requirement to discover those sets of relevant weights in the network for efficient local weight updates. In particular, to determine the significant and insignificant large weights of the baseline model. Weights greater than w_0 that yield a complexity cost closer to 1 require a regularization using the penalty parameter λ. However, we do not predefined the numbers of weights to eliminate, the Algorithm 2 itself will decide this number based on the given DNN architecture and the various other constraints. The regularization considers a scenario where the initialized model produces a higher error value $\mathcal{E}_t$ as in line 7. For better performance, we utilized the set of parameters to produce a lower error value $\mathcal{E}_j$. After this stage, in lines 15 and 16, the estimated computational memory footprints and execution time are compared with that of the initialized values in line 6 to return the minimal memory constraint produced by the client device model. Device models with minimal resource consumption are returned to the coordinating server in Algorithm 3 together with their weights for model averaging. Then, the coordinating server can update the client model weights in a federated setting and performs 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 aggregate model of REFDL. The memory and processor savings for each client device at each federated round and accumulating all these savings can lead to significant savings when the model is converged.

$$R = \lambda \sum_{j=1}^{W} \frac{(w_j^2/w_0^2)}{(1 + w_j^2/w_0^2)} \tag{4}$$

Algorithm 3 Coordination Procedure for Alg. 1 and 2

Server Executes:

1: **function** Server Weights Update
2: initialize weight w;
3: **while** $t \leq n$ **do** ▷ n federated round
4: $R \leftarrow$ random set of $max(C.K, 1)$ ▷ $C.K$ fraction of clients K
5: **for** $k \in R$ in parallel **do** ▷ k client index
6: Weight device update ▷ Federated model weight update for Alg. 1 or 2
7: **end for**
8: Averaged weights update
9: **end while**
10: **return** Averaged updated weights
11: **end function**

4 Evaluation

This section describes the evaluation criteria of the BFDL and REFDL methods. It also presents the datasets used in the evaluation of the proposed approach.

4.1 Utilized Datasets

The N-BaIoT dataset contains various realistic data samples from nine commercial IoT devices that collectively represent multitudes of botnet and benign network traffic flows [25]. Each device is either infected by a variety of BASHLITE or Mirai attacks, with some regular instances. We randomly consider eight devices subsets with the most IoT specification. These devices are a (i) Danmini Doorbell, (ii) Ecoobee Thermostat, (iii) Ennio Doorbell, (iv) Provision PT-737E, (v) Provision PT-838, (vi) Samsung SNH-1011-N, (vii) SimpleHome XCS-1002-WHT, and (ix) SimpleHome XCS-1003-WHT. Each device consists of sufficient records of variational attacks such as ack, syn, scan, junk, tcp, udp, udpplain, combo and regular instances containing a numeric representation of traffic flows with 115 features vector. For this reason, the N-BaIoT dataset serves as a benchmark for the proposal of a device-centric IoT security monitoring mechanism. We utilized mentioned commercial devices subsets data of the N-BaIoT for federated training and testing of BFDL and REFDL models.

The WUSTL dataset consists of multiple flows of traffic from an emulated SCADA system [26]. This dataset can be appropriate for investigating the feasibility of AI algorithms for security monitoring purposes. The raw data consists of 7,037,983 numeric data samples. As a result, we consider the distribution of 471,545 attacks and 6,566,438 normal instances to evaluate our method.

4.2 Virtual Workers Experimental Setup and Implementation

We used Python 3.76 on a desktop computer with Intel Xeon E5-2695(4 core) CPUs running at 2.10 GHz with 16.0 GB installed memory to build each technique. For profiling memory consumption, we utilized the integrated memory usage [27]. We utilized PyTorch version 1.4.0 [28] and PySyft version 0.2.9 [29] frameworks for the virtual on-device training. Pysyft framework simplifies the creation of virtual workers. We utilized these virtual workers to simulate the FL scenario for the BFDL and REFDL. These workers emulate real virtual machines and can run as a separate process within the same python program with their dataset. Our federation training procedure considered four clients' virtual workers and a coordinating server worker receiving the computational updates from each virtual client worker model. Each federated client model consists of an input layer, four hidden layers and an output layer. The topology selection against each dataset utilized [30] to minimize operations and improve the performance metrics. The experimental settings considered are appropriate for binary classification as returned by the [30] parameter tuning technique. The overall architectural settings remain identical for evaluating the BFDL and proposed REFDL technique. Table 1 presents the utilized model architecture of each FL technique against each

dataset. Regarding the Wustl dataset, the selected topology was returned based on [30] tuning technique.

Table 1: Architecture and distribution of normal and attack for each device data.

Device	Normal	Attack	Inputs	Outputs	Architecture
Danmini Doorbell	49,548	968,750	115	1	83-128-128-83
Ecobee Thermostat	13,113	822,763	115	1	83-128-128-83
Ennio Doorbell	39,100	316,400	115	1	83-128-128-83
Provision PT-737E	62,154	766,106	115	1	83-128-128-83
Provision PT-838	98,514	729,862	115	1	83-128-128-83
Samsung SNH-1011-N	52,150	323,072	115	1	83-128-128-83
SimpleHome XCS-1002-WHT	46,585	816,471	115	1	83-128-128-83
SimpleHome XCS-1003-WHT	19,528	831,298	115	1	83-128-128-83
Wustl	6,566,438	471,545	6	1	26-128-128-26

For the baseline and optimized model training procedure, $lr = 0.001$ was utilized. We used 0.01 values for both λ, $\triangle\lambda$ and threshold w_0 [31] with 4 micro-batches to build the model of REFDL method. The activation function considered in the fully connected layers is relu [32] with sigmoid in the output layer. Both BFDL and REFDL use an SGD optimizer appropriate for running FedAvg training. Each federated model was trained in 128 batches within four epochs in 30 worker's communications rounds for optimum convergences. After completing the client's model training, average weight values are sent to the coordinating worker. This worker aggregates those weights to update the global model. Codes for this implementation are made publicly accessible for exploration and reproduction purposes [33].

4.3 Testbed Experimental Setup and Implementation

To test the efficient federated communication of the REFDL against BFDL in a testbed setting, we utilized the PySyft version 0.2.9 [29] python framework over a network with a client and server-class connected via a WebSocket (WS). Since PyTorch is a potential library for PySyft, we utilize it to build an edge computing FL training scenario for resource-constrained devices. The environmental settings mimic the client's server communication scenario in a distributed manner. In this context, it can support the building of simulated and realistic testbed settings. In the network realistic testbed settings, we considered 4 Gigabyte Brix (GB-BXBT-2807) with a laptop (see Figure 1). The personal laptop represents the coordinating server in a wireless network to emulate low-frequency connections. The server is responsible for model weights aggregation and distribution to clients. The client's devices in Alg. 1 and 2 are responsible for local model training using the server model weights on the client's dataset and returning client weights to the server. Therefore, the communication workload is higher at the client-side containing the edge devices than the server machine. The installed Operating

System (OS) in GB-BXBT-2807 clients is Ubuntu version 20.04.4 LTS. Each client contains an installation for the PySyft framework and its dependencies. Federated network testbed implementations codes are publicly accessible [34].

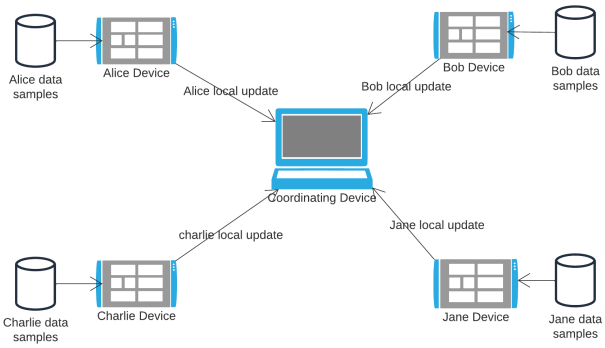

Fig. 1: BFDL and REFDL model training testbed with gigabyte devices.

For evaluating the simulated runtime and real execution time of BFDL and REFDL, experiments with four workers with their distributed training data (Alice, Bob, Charlie and Jane as shown in Figure 1) were performed. A federated communication round of 50 is used, with two epoch iterations, within a 64 mini-batch size as returned by the optimized tuning procedure. The test batch sample size selection is 1000 with $lr = 0.01$ for effective FedAvg SGD training. The utilized real-time models for each federated client of both Algorithm 1 and 2 contain an input layer and four identical hidden layers (128-128-128-128) with an output layer or layers as the case may be. The chosen architecture can support effective and efficient model convergence. To test the REFDL effectiveness and generalizability, we considered the CNN DNN variant in realistic settings with clients utilizing the MNIST image dataset [35]. The CNN architecture contains two convolutional layers (Conv-2). The first 2D convolutional layer requires one input to output 20 convolutional features using a 5 square kernel (1, 20, 5, 1). The second 2D convolutional layer requires 20 input layers to output 50 convolutional features using a 3 square kernel (20, 50, 5, 1). The architecture in the first real-time layer is (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 a dropout utilization. The fully connected hidden layers in the convolutional are similar to the version described in Table 1.

5 Results and Discussion

This section discusses the experimental results. It details the evaluation comparison of the optimized REFDL and baseline BFDL FedAvg models in simulation and testbed settings across datasets.

5.1 Virtual Workers Simulation Results

We investigated the resource consumption for training BFDL and REFDL federated methods with nine utilized IoT datasets on virtual client workers. Table 2 presents the memory and time usage across each dataset. REFDL training procedure produces lower runtime and memory footprints. However, the accuracy for both REFDL and BFDL remained the same across each benchmark dataset. The reason can be the tested datasets are highly imbalanced with large number of testing records and considering the pruning applied in lines 13 and 14 of Algorithm 2, and each model uses a similar network architecture [36]. Refer to Table 4 and Figure 10 for comparison with balanced dataset with minimal number of testing records.

Table 2: Federated model training memory consumption between REFDL and BFDL.

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

Figures 2 and 3 show the percentage of memory and time reduction by REFDL as reflected in Table 2. The results demonstrate a significant percentage of memory saving across each dataset. Regarding client processing runtime, REFDL is more efficient. It indicates less complexity, faster learning capability and effective performance behaviour over BFDL. These resources minimization make it a better choice for IoT security monitoring. Especially for the on-device learning across various distributed resource-constrained edge devices.

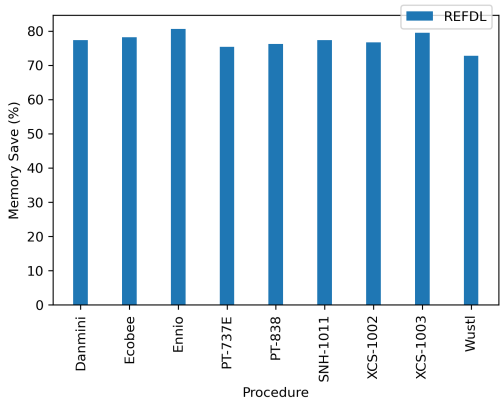

Fig. 2: REFDL federated model training memory resources saved against datasets.

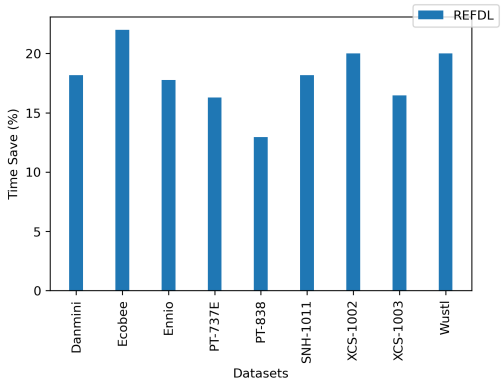

Fig. 3: REFDL federated model training time resources saved against datasets.

The results in Table 3 are for the implemented BFDL method and its optimized counterpart REFDL against training procedures. It compared the training time, memory requirements and accuracy against model hidden layers (L) and virtual workers (VW) variations. As presented, the REFDL requires lower memory and time as tested with the XCS-1003 dataset. Both BFDL and REFDL federated models produce slightly better accuracy with four hidden layers (4L). As a result, the increment of hidden layers influences effective federated learning in distributed settings. However, the clients with higher computational power can influence on the global model significantly than the clients with lower computational resources. The class imbalance across clients can also influence the accuracy of the global model. It would be interesting to investigates these limitations in future work.

The illustration in Figures 4 and 5 present the memory and time savings of REFDL with reference to Table 3. The results illustrate a better resource (memory and time) minimization of REFDL against each training procedure. The results demonstrate REFDL's capability of savings more resources using the

Table 3: Performance comparisons against FL training procedure on SimpleHome XCS-1003 dataset.

Procedure	Model	Memory MB	Time minutes	Test acc %
2VW-3L	BFDL	1.906	0.038	97.72
	REFDL	0.550	0.027	97.72
2VW-4L	BFDL	2.698	0.046	97.73
	REFDL	0.052	0.036	97.73
4VW-3L	BFDL	2.971	0.067	97.72
	REFDL	0.294	0.060	97.72
4VW-4L	BFDL	3.914	0.085	97.73
	REFDL	0.801	0.071	97.73

PySyft virtual worker's constructs that emulate real virtual machines and run as a separate process within the same python program. In particular, it demonstrates the significant memory savings of REFDL with two virtual workers (2VW) and four hidden layers (4L) model architecture. Also, it shows that increments of virtual workers can facilitate better memory savings with three hidden layer network architecture.

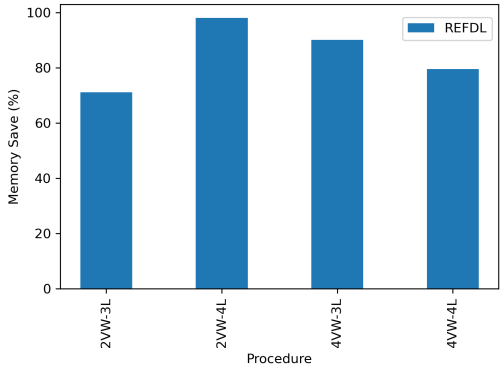

Fig. 4: REFDL federated model training memory resources saved with XCS-1003 dataset.

To test the generalization and effectiveness of REFDL in other domains other than cybersecurity, we examined its performance over the MNIST image dataset with integrated CNN and FCNN in the FL scenario (see Table 4). This is good to assess the method's performance over non-IoT datasets so that it can be a generic solution for on-device learning. In particular, to exploit the resource-saving capability of CNN that offers a promise in image classification. In this aspect, we utilized the PySyft WS (network) simulated workers and examined

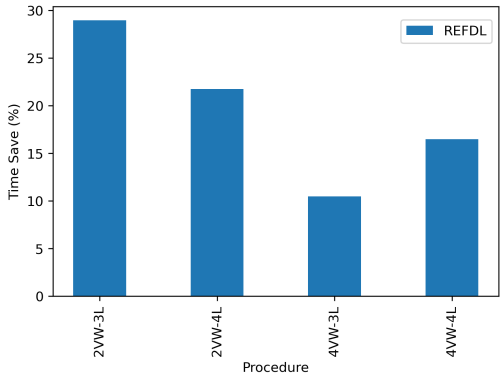

Fig. 5: REFDL federated model training time resources saved with XCS-1003 dataset.

the performance of the BFDL and REFDL techniques in each federated training. This is to assess REFDL performance using a simulated network with a client and server scenario running on the same machine, not like PySft virtual workers counterparts that run as construct within the same python program. With each DNN (CNN and FCNN) variant, the REFDL demonstrates better accuracy than its BFDL counterparts. The reason for better performance with the MNIST dataset can be because the distribution of the dataset is highly balanced for both the training and testing cases. In addition, it produces lower training execution time. These results show the important of regularization [37] and [38] on accuracy against DNN variation. This attracts further investigation in realistic settings.

Table 4: Simulated federated training performance comparison between BFDL and REFDL with MNIST dataset.

Procedure	FL	Time minutes	Time save (%)	Test set acc %
FCNN-MNIST	BFDL	1.393	N/A	34.64
	REFDL	1.346	3.374	91.03
CNN-MNIST	BFDL	1.583	N/A	90.59
	REFDL	1.457	7.960	98.28

5.2 Network Workers Testbed Results

With Ennio Doorbell and Samsung SNH randomly selected IoT datasets, we investigated the memory consumption of training REFDL and BFDL across four GB-BXBT-2807 edge devices over wireless network testbed settings. The

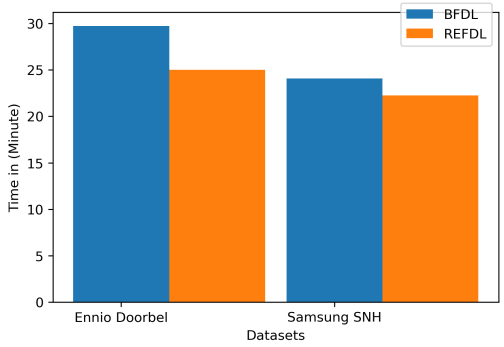

Fig. 6: Federated model training execution time of REFDL and BFDL against datasets.

reported memory values in Table 5 is averaged based on the four devices. The results show that the REFDL can detect IoT attacks with minimal memory than BFDL in real-time. However, it should be noted that the amount of memory and CPU (runtime) savings may vary depending on the number of clients in the federation and the nature of the feature distribution. It is expected that a large number of clients handling complex data in a federation could lead to higher savings.

Table 5: Federated model testbed training memory consumption between REFDL and BFDL.

Dataset	Model	Memory MB	Memory save %	Test acc %
Ennio Doorbell	BFDL	33.965	N/A	89.00
	REFDL	31.981	5.84	89.00
Samsung SNH	BFDL	32.519	N/A	86.10
	REFDL	30.550	6.05	86.10

In Figure 6, we present averaged estimated convergence real-time for training BFDL and REFDL on GB-BXBT-2807 testbed against the Ennio Doorbell and Samsung SNH IoT datasets. The REFDL requires less real-time than BFDL in detecting IoT attacks. The results demonstrate the effectiveness of REFDL in saving computational resources in resource-constrained environments. As illustrated in Figure 7, the savings can be better with many decentralized edge devices. The result demonstrates the savings advantage of REFDL in realistic network settings over the simulated virtual WS connections counterparts. It indicates REFDL's capability in saving more resources in a resource-limited environment with multiple client devices.

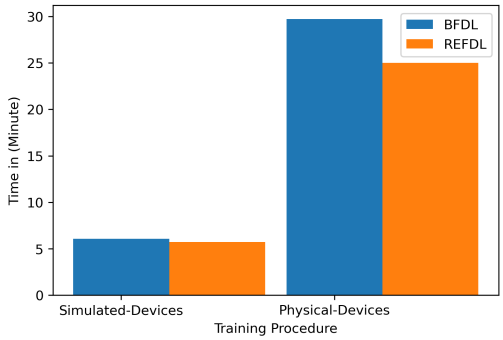

Fig. 7: REFDL and BFDL training execution time in a simulated and realistic network environment against Ennio dataset.

We examined the real-time savings of the REFDL over BFDL against the MNIST dataset. Figure 8 show that REFDL is more efficient than BFDL across each training procedure. The MNIST-CNN federated training procedure is more computationally expensive than the MNIST-FCNN. In this context, the FCNN DNN variant of REFDL can be an appropriate choice for on-device learning if savings resources are the target objectives. In that case, REFDL stands more suitable method for deployment in an IoT resource environment.

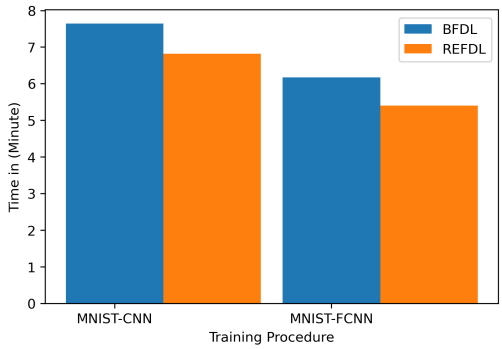

Fig. 8: Federated model training execution time usage between REFDL and BFDL with mnist dataset.

Figure 9 shows the convergence accuracy of REFDL and BFDL against DNN variants with the MNIST dataset. In each case, REFDL stands to be a better model than BFDL. It can classify image samples accurately with integrated CNN and FCNN (DNN) model variants. The result suggests the advantage of optimization mechanisms in producing a global deep federated model. It further demonstrates the effectiveness of integrating CNN in the FL method to improve

accuracy performance. This is good as it leverages the tradeoff between each DNN model during on-device learning.

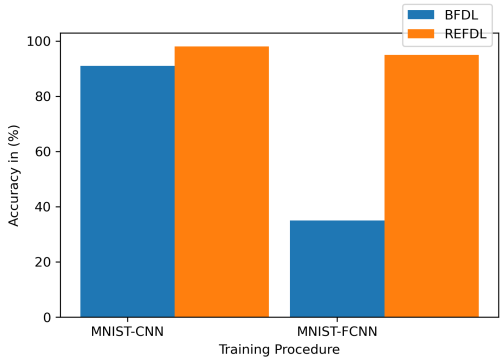

Fig. 9: Federated model accuracy comparison between REFDL and BFDL with mnist dataset.

To test the effectiveness and faster learning of REFDL on GB-BXBT-2807 testbed federated settings, we vary the epoch iterations using the FCNN-MNIST procedure. In that context, we can assess the performance of each federated method in real-time. As shown in Figure 10, the REFDL can achieve a better accuracy even with one local epoch and 50 communication round. This trends of providing higher accuracy remain stable across each epoch iteration. The result demonstrates REFDL appropriateness and faster learning capability across edge devices, especially with the integrated FCNN model. REFDL minimum number of epoch requirements is advantageous. Especially in an environmental setting such as IoT with inherited limited memory resources.

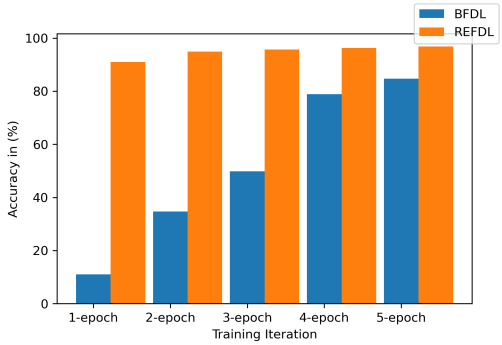

Fig. 10: Federated model accuracy performance with epochs between REFDL and BFDL against mnist dataset.

6 Conclusion

As FL uses a distributed ML to enable on-device learning in decentralized edge devices over a network with the support of data privacy across multiple clients, this paper investigated the feasibility of running FL training in resource-constrained environments such as IoT. In particular, to develop feasible and effective security solutions for IoT devices. In this paper, we utilized FedAvg (BFDL algorithm) with carefully selected model optimization techniques to produce an effective and resource-efficient REFDL federated model. The experiments evaluation with eight IoT datasets and one image dataset in simulated and GB-BXBT-2807 realistic testbed settings demonstrate the effectiveness, low complexity and efficient nature of REFDL. It detects IoT attacks accurately using minimal resources than its counterparts. Also, it can perform better in classifying image samples with fully connected and convolutional deep neural network models in a federated training scenario. In addition, REFDL requires fewer epochs to produce a more accurate FL model than its counterparts. These motivational results attract further investigation for utilizing more computational networks nodes/client devices at deployment, particularly over-wired and wireless settings using our testbed. In addition, we plan to investigate the resilient capability of the REFDL to enhance its security robustness against adversarial attacks in a realistic network setting with various connected edge devices other than the ones considered in this paper. This can enable us to examine the resource efficiency and security monitoring performance of our proposed method capability and potentiality across multiple decentralized edge devices.

7 Acknowledgment

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

References

1. J. Howell, "Number of connected iot devices will surge to 125 billion by 2030, ihs markit says," [online]. Available: https://news.ihsmarkit.com/prviewer/release_only/slug/number-connected-iot-devices-will-surge-125-billion-2030-ihs-markit-says, 2017, [online; accessed 17-August-2018].
2. M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th {USENIX} security symposium ({USENIX} Security 17)*, 2017, pp. 1093–1110.
3. I. V. Kotenko, I. Saenko, and A. Branitskiy, "Applying big data processing and machine learning methods for mobile internet of things security monitoring." *J. Internet Serv. Inf. Secur.*, vol. 8, no. 3, pp. 54–63, 2018.
4. J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

5. Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.
6. M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.
7. X. Li, H. Liu, W. Wang, Y. Zheng, H. Lv, and Z. Lv, "Big data analysis of the internet of things in the digital twins of smart city based on deep learning," *Future Generation Computer Systems*, vol. 128, pp. 167–177, 2022.
8. S. Shen, R. Li, Z. Zhao, Q. Liu, J. Liang, and H. Zhang, "Efficient deep structure learning for resource-limited iot devices," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.
9. J. Rock, W. Roth, M. Toth, P. Meissner, and F. Pernkopf, "Resource-efficient deep neural networks for automotive radar interference mitigation," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 4, pp. 927–940, 2021.
10. S. Kodali, P. Hansen, N. Mulholland, P. Whatmough, D. Brooks, and G.-Y. Wei, "Applications of deep neural networks for ultra low power iot," in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 589–592.
11. I. Zakariyya, H. Kalutarage, and M. O. Al-Kadri, "Robust, effective and resource efficient deep neural network for intrusion detection in iot networks," in *Proceedings of the 8th ACM on Cyber-Physical System Security Workshop*, 2022, pp. 41–51.
12. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.
13. D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor, "Chained anomaly detection models for federated learning: An intrusion detection case study," *Applied Sciences*, vol. 8, no. 12, p. 2663, 2018.
14. A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A survey on federated learning for resource-constrained iot devices," *IEEE Internet of Things Journal*, 2021.
15. T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "Dïot: A federated self-learning anomaly detection system for iot," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 756–767.
16. Y. Liu, N. Kumar, Z. Xiong, W. Y. B. Lim, J. Kang, and D. Niyato, "Communication-efficient federated learning for anomaly detection in industrial internet of things," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.
17. Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," *arXiv preprint arXiv:1909.12326*, 2019.
18. K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.
19. S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, and O. Jogunola, "Federated deep learning for zero-day botnet attack detection in iot edge devices," *IEEE Internet of Things Journal*, 2021.
20. I. Zakariyya, H. Kalutarage, and M. O. Al-Kadri, "Memory efficient federated deep learning for intrusion detection in iot networks." CEUR Workshop Proceedings, 2021.

21. Y. Chauvin and D. E. Rumelhart, *Backpropagation: theory, architectures, and applications*. Psychology press, 2013.
22. O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.
23. Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, pp. 103–112, 2019.
24. S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626*, 2015.
25. Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot—network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.
26. M. A. Teixeira, T. Salman, M. Zolanvari, R. Jain, N. Meskin, and M. Samaka, "Scada system testbed for cybersecurity research using machine learning approach," *Future Internet*, vol. 10, no. 8, p. 76, 2018.
27. F. Pedregosa and P. Gervais, "Memory profiler (python)," *Python Software Foundation, https://pypi. org/project/memory-profiler/. Accessed March*, vol. 25, 2019.
28. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.
29. T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017*, 2018.
30. B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn," in *Automated Machine Learning*. Springer, Cham, 2019, pp. 97–111.
31. A. Bosman, A. Engelbrecht, and M. Helbig, "Fitness landscape analysis of weight-elimination neural networks," *Neural Processing Letters*, vol. 48, no. 1, pp. 353–373, 2018.
32. H. Ide and T. Kurita, "Improvement of learning for cnn with relu activation by sparse regularization," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 2684–2691.
33. Zakariyya, "Resource efficient federated algorithm with virtual workers." 2022. [Online]. Available: https://github.com/izakariyya/sim-virtual-fed-dnn
34. I. Zakariyya, "Resource efficient federated algorithm with realistic workers." 2022. [Online]. Available: https://github.com/izakariyya/testbd-fl-iot
35. L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.
36. V. García, R. A. Mollineda, and J. S. Sánchez, "Theoretical analysis of a performance measure for imbalanced data," in *2010 20th International Conference on Pattern Recognition*. IEEE, 2010, pp. 617–620.
37. D. Krueger and R. Memisevic, "Regularizing rnns by stabilizing activations," *arXiv preprint arXiv:1511.08400*, 2015.
38. J. Lever, M. Krzywinski, and N. Altman, "Points of significance: Regularization," *Nature methods*, vol. 13, no. 10, pp. 803–805, 2016.