University of Technology Sydney
Faculty of Engineering and Information Technology
School of Electrical and Data Engineering

Federated Learning for Cyberattack Detection in Decentralized Networks

Viet Khoa Tran

A Thesis Submitted
in Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

under the supervision of

A/Prof. Hoang Dinh, Principal supervisor
A/Prof. Diep N. Nguyen, Co-supervisor
Prof. Eryk Dutkiewicz, Co-supervisor
A/Prof. Nguyen Linh Trung, External supervisor
A/Prof. Nguyen Viet Ha, External supervisor

Sydney, Australia

January 2024

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, VIET KHOA TRAN, declare that this thesis is submitted in fulfilment of the requirements for the award of DOCTOR OF PHILOSOPHY in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used in the thesis are indicated.

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of the requirements for a degree at any other academic institution except as fully acknowledged within the text.

This research is supported by the Australian Government Research Training Program.

Student Name: Viet Khoa Tran

Student Signature: Production Note:
Signature removed prior to publication.

Date: April 15, 2024

ABSTRACT

Federated Learning for Cyberattack Detection in Decentralized Networks

by

Viet Khoa Tran

Recently, the rapid development of various technologies, such as blockchain and Internet-of-Things (IoT), has enabled numerous applications to become integral to many aspects of our daily lives. However, this leads to a massive amount of data and raises serious security concerns. Machine Learning (ML), especially Deep Learning (DL), has been widely used in cyberattack detection to detect cyberattacks in emerging networks. Nevertheless, DL-based cyberattack detection systems usually require a huge amount of data from users/systems. This threatens user privacy because sensitive data may be sent over the network to a centralized server for processing. Additionally, transmitting such a large amount of data imposes communication overhead on the network.

This thesis aims to develop ML frameworks that can efficiently detect cyberattacks/intrusions in decentralized networks, such as IoT and blockchain networks, without exposing their local data over the network. In the first study, we develop a collaborative learning framework that enables a *target network* with unlabeled data to learn "knowledge" from a *source network* with abundant labeled data in IoT networks. Our proposed framework can exchange the learned "knowledge" among various DL models, even when their datasets have different features. The experiments showed that our proposed framework could achieve higher accuracy than state-of-the-art DL-based approaches.

In the second study, we develop a cyberattack detection framework to detect cyberattacks in the network traffic of blockchain networks. Specifically, we first

implement a blockchain network in our laboratory. This blockchain network will serve to generate real traffic data and implement real-time experiments to evaluate the performance of our frameworks. We then propose a collaborative learning model that allows efficient deployment in the blockchain network to detect attacks. Both simulated and real-time experiments have been provided to demonstrate the efficiency of our proposed framework.

In the third study, we propose a collaborative learning framework to detect attacks on blockchain transactions and smart contracts. Our framework can classify various types of blockchain attacks, including intricate attacks at the machine code level which typically necessitate significant time and security expertise. Additionally, our framework enables real-time detection of diverse attack types at Ethereum nodes. The simulated and real-time experiments demonstrate the outperformance of our proposed framework compared to conventional DL models.

All the results above have demonstrated that our proposed federated learning-based frameworks can efficiently be deployed in decentralized networks to detect cyberattacks with high accuracy.

Dedication

Dedicated to my beloved father, my mother, my wife, and my daughters, who have been my unwavering support throughout this journey.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisors – professors Hoang Dinh, Nguyen Linh Trung, Diep N. Nguyen, Eryk Dutkiewicz, and Nguyen Viet Ha – for their guidance, encouragement and support. Without them, this dissertation would have been impossible. During my study, they not only guided me to pursue great and impactful research but also gave valuable advice for my career. Their guidance and support go far beyond this thesis, and I have been lucky to be supervised by them.

I would like to thank all my colleagues and friends at the University of Technology Sydney (UTS), and Advanced Institute of Engineering and Technology (AVITECH), University of Engineering and Technology, Vietnam National University, Hanoi (VNU-UET) for their support, discussion, and friendship. Thanks should also go to the SEDE admin team for handling all the paperwork and forms during my PhD study. I would like to thank the Joint Technology and Innovation Research Centre (JTIRC) between UTS and VNU-UET, and the Faculty of Engineering and Information Technology (FEIT) for giving me the opportunities and financial support during the past years.

This work was partly supported by the ASEAN IVO projects on "Cyber-Attack Detection and Information Security for Industry 4.0" and on "Agricultural IoT based on Edge computing".

I would especially like to thank my family and friends for their endless love and support that gave me the strength to overcome difficulties in my life.

Contents

List of Publications

Journal Papers

J-1. **T. V. Khoa**, D. T. Hoang, N. L. Trung, C. T. Nguyen, T. T. Quynh, D. N. Nguyen, N. V. Ha and E. Dutkiewicz, "Deep Transfer Learning: A Novel Collaborative Learning Model for Cyberattack Detection Systems in IoT Networks," *IEEE Internet of Things Journal*, vol. 10, no. 10, pp. 8578-8589, 15 May 2023. (*Corresponding to Chapter 2*).

J-2. **T. V. Khoa**, D. H. Son, D. T. Hoang, N. L. Trung, T. T. T. Quynh, D. N. Nguyen, N. V. Ha, and E. Dutkiewicz, "Collaborative Learning for Cyberattack Detection in Blockchain Networks," *IEEE Transactions on Systems, Man and Cybernetics: Systems*, early access, 03 April 2024. (*Corresponding to Chapter 3*).

J-3. **T. V. Khoa**, D. H. Son, Chi-Hieu Nguyen, D. T. Hoang, N. L. Trung, D. N. Nguyen, T. T. T. Quynh, Trong-Minh Hoang, N. V. Ha, and E. Dutkiewicz, "Securing Blockchain Systems: A Novel Collaborative Learning Framework to Detect Attacks in Transactions and Smart Contracts," *IEEE Transactions on Systems, Man and Cybernetics: Systems*, under review. (*Corresponding to Chapter 4*).

J-4. C. T. Nguyen, Y. M. Saputra, N. V. Huynh, N. T. Nguyen, **T. V. Khoa**, B. M. Tuan, D. N. Nguyen, D. T. Hoang, T. X. Vu, E. Dutkiewicz, S. Chatzinotas and B. Ottersten, "A Comprehensive Survey of Enabling and Emerging Technologies for Social Distancing—Part I: Fundamentals and Enabling Technologies," *IEEE Access*, vol. 8, pp. 153479-153507, Aug. 2020.

J-5. C. T. Nguyen, Y. M. Saputra, N. V. Huynh, N. T. Nguyen, **T. V. Khoa**, B.

M. Tuan, D. N. Nguyen, D. T. Hoang, T. X. Vu, E. Dutkiewicz, S. Chatzinotas and B. Ottersten, "A Comprehensive Survey of Enabling and Emerging Technologies for Social Distancing—Part II: Emerging Technologies and Open Issues," *IEEE Access*, vol. 8, pp. 154209-154236, Aug. 2020.

Conference Papers

C-1. **T. V. Khoa**, D. H. Son, D. T. Hoang, N. L. Trung, T. T. T. Quynh, D. N. Nguyen, N. V. Ha, and E. Dutkiewicz, "Real-time Cyberattack Detection with Collaborative Learning for Blockchain Networks," *2024 IEEE Wireless Communications and Networking Conference (WCNC)*, accepted.

C-2. D. H. Son, T. T. T. Quynh, **T. V. Khoa**, D. T. Hoang, N. L. Trung, N. V. Ha, D. Niyato, D. N. Nguyen, and E. Dutkiewicz, "An Effective Framework of Private Ethereum Blockchain Networks for Smart Grid," *2021 International Conference on Advanced Technologies for Communications (ATC)*, 2021, pp. 312-317, [**Best student paper award**].

C-3. **T. V. Khoa**, Y. M. Saputra, D. T. Hoang, N. L. Trung, D. N. Nguyen, N. V. Ha, and E. Dutkiewicz, "Collaborative Learning Model for Cyberattack Detection Systems in IoT Industry 4.0," *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, 2020, pp. 1-6.

List of Figures

Abbreviation

IoT	Internet of Things
AI	Artificial Intelligence
6G	Sixth-generation Wireless Network
IDS	Intrusion Detection System
ML	Machine Learning
DL	Deep Learning
UDL	Unsupervised Deep Learning
FL	Federated Learning
TL	Transfer Learning
DNN	Deep Neural Network
DBN	Deep Belief Network
GRBM	Gaussian Restricted Boltzmann Machines
RBM	Restricted Boltzmann Machine
CNN	Convolutional Neural Network
Co-CNN	Collaborative Deep Convolutional Neural Network
MN	Mining Node
EN	Ethereum Node
SC	Smart Contract
CS	Centralized Server
BNAT	Blockchain Network Attack Traffic
BCID	Blockchain Intrusion Detection
ABTD	Attacks on Blockchain Transactions Dataset
BCEC	Blockchain Code Extraction and Conversion Tool

Chapter 1

Motivation, Background, and Literature Review

In this chapter, we first present the motivation of this thesis. We then provide the background of deep learning, federated learning, and transfer learning. After that, we discuss the state of the art, challenges, and solutions of applying these techniques to intrusion detection systems in decentralized networks. Finally, we highlight the structure and the main contributions of this thesis.

1.1 Motivation

In recent years, the rapid development of various technologies, such as 5G/6G, Industry 4.0, and Internet-of-Things (IoT), has led to the integration of numerous applications into various aspects of our daily lives. However, such fast growth has also led to an unprecedented massive amount of data and the proliferation of interconnected devices, e.g., sensors, smart cars, and cameras, which raises serious security and privacy concerns. Particularly, the increasing number of emerging applications has also brought forth various new types of cyberattacks. For example, the number of new (zero-day) cyberattacks has increased by 60% from 2018 to 2019 [1]. Beside the dire consequences to the economy, e.g., ransomware alone cost more than $5 billion globally in 2017 [2], cyberattacks pose serious threats to other areas with highly sensitive information such as healthcare and public security. As a result, cyberattack detection methods play a key role in detecting and promptly preventing the consequences of cyberattacks in future IoT networks.

Beside IoT, Blockchain also has been emerging as a novel technology in stor-

ing and managing data with many advantages over conventional data management systems. In particular, unlike traditional centralized data management solutions, blockchain technology enables data to be stored in a distributed manner across multiple nodes. In this way, data can be accessed and processed simultaneously at multiple nodes, thus avoiding the problem of bottlenecks and single point of failure. More importantly, one of the most important features of blockchain technology is to enable data to be stored in blocks, and once a block of data is verified and placed in the chain, it cannot be modified and/or deleted. In this way, the data's integrity can be protected thanks to outstanding features of blockchain, e.g., decentralization, immutability, auditability, and fault tolerance [3]. As a result, there are more and more applications of blockchain technology in our lives including finance, healthcare, logistics, and IoT systems [3–6]. Due to the rapid success with a wide range of applications in most areas, especially in money transfer and cryptocurrency, blockchain-based systems have been becoming targets of many new-generation cyberattacks. For example, in September 2020, KuCoin, a crypto exchange based in Singapore, announced that its system was hacked and the hackers stole over $281 million worth of coins and tokens [7]. Most recently, in January 2022, Chainalysis reported that North Korean hackers performed seven attacks on cryptocurrency platforms and stole nearly $400 million from digital assets in 2021 [8]. Although most current attacks target on virtual money exchange systems, a number of blockchain applications in critical areas such as healthcare [9] and food supply chains [10] could be potential for attackers in the near future. If these attacks happen, they not only cause huge losses on our assets but can also lead to many serious issues related to human health and lives. Therefore, solutions to detect and prevent attacks in blockchain networks are becoming more urgent than ever.

Cyberattack detection methods can be classified into signature-based and anomaly-based methods. Signature-based methods rely on the prior knowledge (signatures)

of a cyberattack to detect its perpetration. Although such methods can achieve a low false-positive rate, they require frequent updates of the signature database to achieve high performance. Moreover, these methods cannot detect various types of attacks, such as zero-day attacks, because their signatures are yet unknown to the database or defense systems. In contrast, anomaly-based methods can detect attacks based on the anomalies in the incoming traffic. Although these methods may cause more false alarms as compared to those of the signature-based methods, anomaly-based methods are more effective in detecting new types of cyberattacks because they do not rely on known signatures. Since both signature-based and anomaly-based methods have complementary advantages, it is desirable to have solutions that can leverage the advantages of both techniques, but at the same time can effectively overcome the current limitations of these techniques.

Recently, with outstanding classification ability, Machine Learning (ML) techniques, especially deep learning (DL), have been widely applied for cyberattack detection problems. Particularly, DL models can effectively learn the signatures of various cyberattack types. Moreover, DL models even can detect new types of attacks that have never been learned/trained before [11]. As a result, in addition to the ability to effectively detect various types of known attacks, DL-based methods can be also used to identify new types of cyberattacks even without requiring prior knowledge of the attack signatures. Nonetheless, DL-based cyberattack detection systems are also facing numerous practical challenges. Particularly, conventional DL approaches usually require a huge amount of data to achieve high performance. However, in numerous applications, data are very difficult to collect because they are often stored locally on user devices such as IoT devices, smartphones, and wearable devices. This poses a threat to user privacy because sensitive data (e.g., location and private information) have to be sent over the network and stored at the centralized server for processing. Besides privacy concerns, transmitting such a collectively large

amount of data also imposes an extra communication burden on the network. Consequently, these limitations have been hindering the effectiveness of DL techniques in cyberattack detection systems.

To address these problems, Federated Learning (FL) has emerged to be a highly effective solution. Unlike conventional DL techniques that collect data and train the global model at a central server, FL enables the learning process to be distributed across all devices. Particularly, instead of sending data to a central server, the local data can be used to train the global model locally on each user device. Then, the obtained model weights of each device are periodically sent to a central server for aggregation. Afterward, the aggregated weights are sent back to all devices to update the weights of the local models. Since only the weights are exchanged in FL, both the privacy and communication overhead issues can be mitigated [12].

1.2 Background

In this section, we present the fundamental knowledge of deep learning, federated learning, and transfer learning. They are the key techniques that will be used in the following chapters of the thesis.

1.2.1 Deep Learning

Deep learning – a sub-field of machine learning – is a multilayer Deep Neuron Network (DNN) that imitates the neural network of human brains. Figure 1.1 describes a multilayer neural network. In this network, each neuron connects to others by a hypothesis function that is characterized by a matrix of weights $\mathbf{W}$ as the following [13]:

$$\mathbf{Z} = \mathbf{W}\mathbf{X} + \mathbf{b}, \tag{1.1}$$

where $\mathbf{X}$ is a vector of the input data, $\mathbf{Z}$ is a vector of output data of the neural network, and $\mathbf{b}$ is the vector of bias values. Typically, the deep neural network

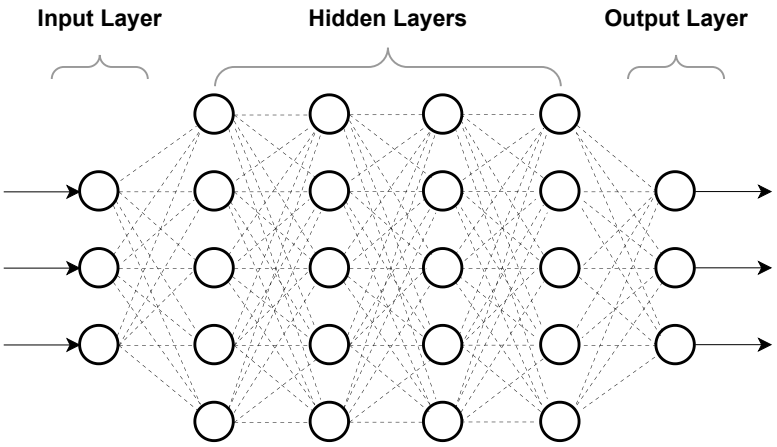


Figure 1.1: Multilayer deep neural network.

architecture mainly consists of three layers including an input layer, hidden layers, and an output layer [13]. The input layer is the gateway to process and transform the input data into the space of the hidden layers. The hidden layers, the core of DL, include multiple neuron layers to process, analyze and optimize the input data. The output layer is the last layer which represents the results of the whole DL process.

From the processing perspective, DL includes two main processes namely training process and testing process. The training process is the main process of DL to analyze and process the input data. First, forward propagation is used to calculate the vector of predicted values. Then, the vector of predicted values $\mathbf{Z}$ (i.e., the output of the neural network) is evaluated to calculate the errors by the loss function. Here, the loss function depends on the type of output or function, e.g., the logistic loss function [14] with a predicted value z and a label y:

$$J(z, y) = \log\big(1 + \exp(-zy)\big), \tag{1.2}$$

The output of the loss function is used by the backpropagation to update the parameters of the neural network. The forward propagation and backpropagation are repeated in the training process to optimize the output and neural network. After

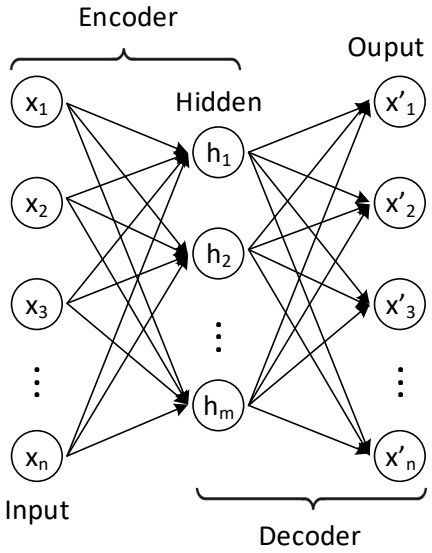


Figure 1.2: Architecture of a basic AE.

the training process, the testing process is performed to evaluate the efficiency of the learning process [13].

1.2.1.1 Autoencoder

Autoencoder (AE) is a type of unsupervised deep learning model that can effectively learn the latent feature representation of the input data for multiple purposes, such as dimensionality reduction and generative modeling [15]. As illustrated in Figure 1.2, a basic AE consists of three layers, i.e., one input layer, one hidden layer, and one output layer. Note that, for deep AEs, we can use multiple hidden layers to improve the learning performance. The input and hidden layers together form the encoder which is used to map the vector of the input data $\mathbf{X}$ with n-features into the representation of another vector with m-features ($\mathbf{H}$) by the encode function $f(\cdot)$ as $\mathbf{H} = f(\mathbf{X})$. Then, the decoder, i.e., the other hidden layers and the output layer, uses $\mathbf{H}$ to reconstruct the vector of input data by the decode function $g(\cdot)$. This process creates the vector of output data $\mathbf{X}_d$ as $\mathbf{X}_d = g(\mathbf{H})$. During the training process, the AE aims to learn the functions $f(\cdot)$ and $g(\cdot)$ that minimize the

reconstruction error [15], i.e.,

$$\min_{f,g} \|\mathbf{X} - g(f(\mathbf{X}))\|. \tag{1.3}$$

In practice, AE is often employed for dimensionality reduction [15]. The AE is trained with the original data to minimize the reconstruction error, thereby allowing the original data to be reconstructed using the hidden layers with minimal error. After sufficient training, the hidden layers of the encoder become an accurate m-feature representation of the original n-features input data [16]. AE is a very useful mechanism to convert the input data into the feature space as well as to perform unsupervised learning so as to analyze the unlabeled data of the input dataset. In this situation, various ML algorithms, e.g., k-means, can be used to cluster the unlabeled data into different groups. As a result, the AE has been widely used for various purposes, especially in cybersecurity for intrusion detection [17, 18].

1.2.1.2 Deep Belief Network

The Deep Belief Network (DBN) is a type of deep neural network that is used as a generative model for both labeled and unlabeled data. Therefore, unlike other supervised deep neural networks which use labeled data to train the neural networks (e.g., convolutional neural networks [13]), a DBN has two stages in the training process. The first stage is the pre-training process where the DBN is trained using an unlabeled dataset. The second stage is the fine-tuning process where the DBN is trained using a labeled dataset. Thereby, the DBN can better represent the characteristics of the labeled dataset, and thus it can classify the normal behavior and different types of attacks with high accuracies. In addition, the DBN includes multiple Restricted Boltzmann Machine (RBM) layers for latent representation [19]. In the DBN training process, the current layer generates a latent representation by

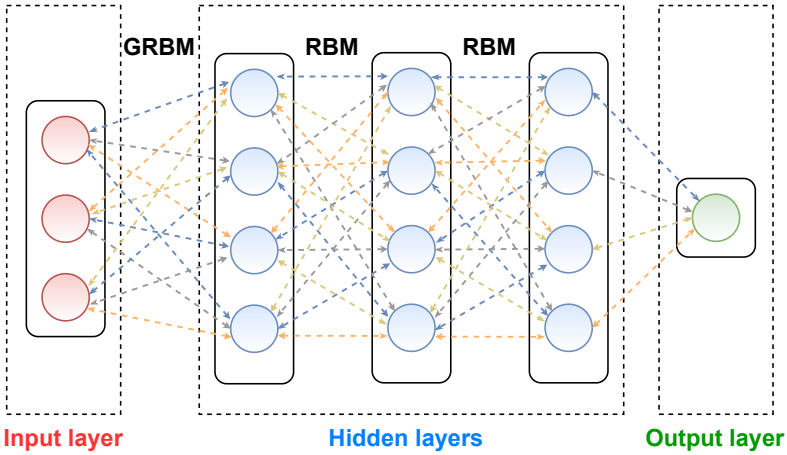


Figure 1.3: Deep Belief Network structure.

using the latent representation of the previous layer as the input. Unlike other deep neural networks which can also process both labeled and unlabeled data (e.g., autoencoder deep learning network [13]), the DBN optimizes the energy function of each layer to have better a latent representation of data on each RBM layer in each iteration. Therefore, the DBN is more appropriate to analyze the network traffic where the samples and features have relative coherence with each other.

The whole process of DBN is illustrated in Figure 1.3. Like other DNNs, the structure of DBN has three components: an input layer, an output layer, and multiple hidden layers. As can be seen in Figure 1.3, the Gaussian Restricted Boltzmann Machine (GRBM) layer – a type of RBM that can process real values of data – is the input layer to receive and transform the input data into binary values. We denote by $\boldsymbol{v}$ and $\boldsymbol{h}$ the vectors of visible and hidden layers of the DBN, respectively. We denote by M and N the numbers of visible and hidden neurons of GRBM. In addition, we denote by h_n, $h_n \in \boldsymbol{h}$, and v_m, $v_m \in \boldsymbol{v}$ the hidden layer-n and visible layer-m of the DBN, respectively. As defined in [19], the energy function of GRBM of the DBN is calculated as follows:

$$E_G(\boldsymbol{v}, \boldsymbol{h}) = \sum_{m=1}^{M} \frac{(v_m - b_{1,m})^2}{2\epsilon_m^2} - \sum_{m=1}^{M}\sum_{n=1}^{N} w_{m,n} h_n \frac{v_m}{\epsilon_m} - \sum_{n=1}^{N} b_{2,n} h_n, \tag{1.4}$$

where $w_{m,n}$ is the weight between visible and hidden neurons; $b_{1,m}$ and $b_{2,n}$ indicate the bias of visible and hidden neurons, respectively; and ϵ_m represents the standard deviation of the neuron in the visible layer. From the result of equation (1.4), we can find the probability that is used in the visible layer of GRBM [19] as follows:

$$p_G(\boldsymbol{v}) = \frac{\sum_{\boldsymbol{h}} e^{-E_G(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{v}}\sum_{\boldsymbol{h}} e^{-E_G(\boldsymbol{v},\boldsymbol{h})}}. \tag{1.5}$$

Then, we use the probability in equation (1.5) to calculate the gradients of each GRBM layer with the expectation value $\left\langle . \right\rangle$ as follows [19]:

$$\begin{aligned} \nabla g_{G,m,n} &= \frac{\partial \log p_G(\boldsymbol{v})}{\partial w_{m,n}} \\ &= \left\langle \frac{1}{\epsilon_m} v_m h_n \right\rangle_{dataset} - \left\langle \frac{1}{\epsilon_m} v_m h_n \right\rangle_{model}. \end{aligned} \tag{1.6}$$

Next, the gradient of GRBM layers can be calculated:

$$\nabla g^G = \sum_{m=1}^{M}\sum_{n=1}^{N} \nabla g_{G,m,n}. \tag{1.7}$$

In the next stage, we need to calculate the energy function and the gradient of RBM layers. We denote by M' and N' the numbers of visible and hidden neurons of RBM layers. As defined in [19], the energy functions of RBM layers of the DBN are defined as follows:

$$E_{RBM}(\boldsymbol{v},\boldsymbol{h}) = -\sum_{m=1}^{M'} b_{1,m} v_m - \sum_{m=1}^{M'}\sum_{n=1}^{N'} w_{m,n} v_m h_n - \sum_{n=1}^{N'} b_{2,n} h_n. \tag{1.8}$$

Similar to the GRBM layers, we can calculate the gradient of each RBM layer as follows:

$$\nabla g_{R,m,n} = \left\langle v_m h_n \right\rangle_{dataset} - \left\langle v_m h_n \right\rangle_{model}. \tag{1.9}$$

In addition, the gradient of RBM layers in the DBN can be defined as follows:

$$\nabla g^R = \sum_{m=1}^{M'} \sum_{n=1}^{N'} \nabla g_{R,m,n}. \tag{1.10}$$

After learning with multiple GRBM and RBM layers, we define $\boldsymbol{X}^{g,r}$ as the output of the last hidden layer. Here, the output layer utilizes the softmax regression function to classify the data samples based on probability. We denote by $\boldsymbol{W}^o$ and $\boldsymbol{b}^o$ the weight matrix and bias vector between the output and the last hidden layer, respectively. We then can define the probability of the output Z belonging to Class-t as follows:

$$p^o(Z = t|\boldsymbol{X}^{g,r}, \boldsymbol{W}^o, \boldsymbol{b}^o) = softmax(\boldsymbol{W}^o, \boldsymbol{b}^o), \tag{1.11}$$

where $t \in \{1, \ldots, T\}$ is a class of the output, and T refers to the total number of classes (including different types of attacks and normal behavior). The prediction $\boldsymbol{Z}$ of the probability p^o can be calculated:

$$\boldsymbol{Z} = \underset{t}{\operatorname{argmax}}[p^o(Z = t|\boldsymbol{X}^{g,r}, \boldsymbol{W}^o, \boldsymbol{b}^o)], \tag{1.12}$$

where Z is the output prediction. Then, we can calculate the gradient between the output layer and the last hidden layer from equation (1.11) as follows:

$$\nabla g^o = \frac{\partial p^o(Z = t|\boldsymbol{X}^{g,r}, \boldsymbol{W}^o, \boldsymbol{b}^o)}{\partial \boldsymbol{W}^o}. \tag{1.13}$$

After that, the results of equation (1.7), equation (1.10), and equation (1.13) are used to calculate the total gradient ∇g^t of DBN with multiple GRBM, RBM layers and the output layer of the DBN as follows:

$$\nabla g^t = \nabla g^G + \nabla g^R + \nabla g^o. \tag{1.14}$$

In the training process, the DBN first trains its neural network with unlabeled data for pre-training. Then, DBN uses its labeled data to fine-tune its neural network.

1.2.1.3 Convolutional Neural Network

Convolutional Neural Network (CNN) is a framework that can classify a large amount of labeled data, especially in image classification with high accuracy [20]. The architecture of CNN includes three types of layers, i.e., convolution layer, max pooling layer, and fully connected layer [20]. Figure 1.4 describes the different layers of a CNN. These layers can be described as follows:

- **Convolution layer:** The neurons in this layer are formed in feature maps to learn the feature representation of the input. In addition, these feature maps can connect with others of the previous layer by weight parameters called filter banks [21]. In this layer, the input data is convoluted with weight parameters in every iteration to create feature maps.
- **Max pooling layer:** The main purpose of this layer is to reduce the resolution of feature maps in the previous layer. To do this, this layer selects the largest values in areas of feature map [20] and then sends them to the next layer.
- **Fully connected layer:** This layer performs classification functions for the neural network. In this layer, the feature maps from previous layers are first flattened. They are then put into a fully connected layer for classification. The softmax function is included at the end of this layer to produce the output prediction.

We denote by $\boldsymbol{D}$ a dataset to train a CNN. $\boldsymbol{D}$ includes $\boldsymbol{S}$ images and $\boldsymbol{Y}$ labels so we can denote $\boldsymbol{D} = (\boldsymbol{S}, \boldsymbol{Y})$. We denote by N the number of training layers of

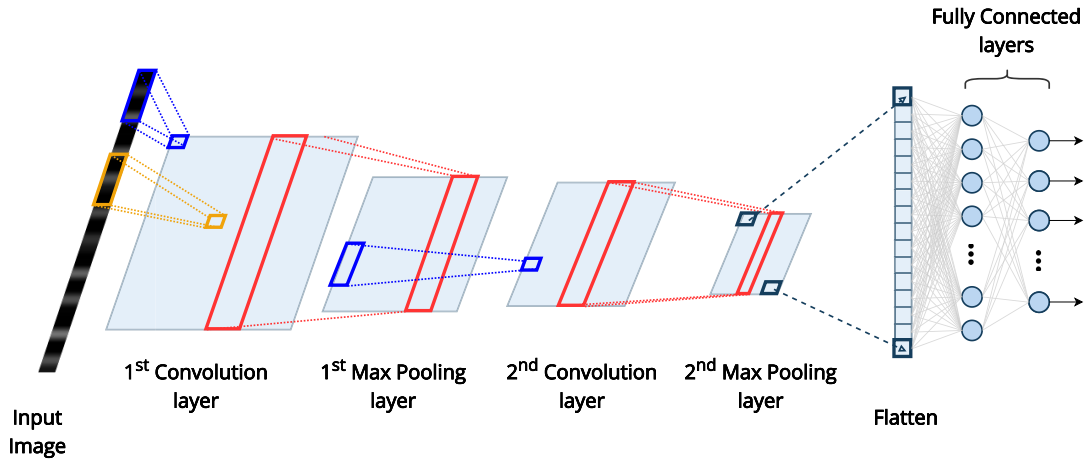


Figure 1.4: The architecture of a CNN model. The convolution layer learns the feature representation of the input. Each max pooling layer reduces the resolution of the feature map in the previous layer. The fully connected layer performs classification functions to produce output [22].

the neural network. We denote by $\boldsymbol{I}$ the matrix features of image $\boldsymbol{S}$, and $\boldsymbol{I_i}$ as the matrix features of image $\boldsymbol{S}$ at iteration i. The output of a convolution layer n, $n \in \{1, \ldots, N\}$, at iteration $i+1$ can be calculated as follows [23]:

$$\boldsymbol{I}_{n+1,i} = \gamma_n\Big(\boldsymbol{I}_{n,i} * \boldsymbol{F}_n\Big), \tag{1.15}$$

where $(*)$ is the convolutional operator, γ_n is the activation function and $\boldsymbol{F}_n$ is the filter bank of layer n. After that, the output of the convolution layer is put into a max pooling layer. The output of a max pooling layer can be calculated as follows:

$$\boldsymbol{I}_{n+2,i} = \varphi\Big(\boldsymbol{I}_{n+1,i}\Big), \tag{1.16}$$

where φ is the max pooling function that selects the maximum value in a pooling area. We denote by $\boldsymbol{I}_{e,i}$ the matrix features of the last data after processing with multiple convolution layers and max pooling layers. $\boldsymbol{I}_{e,i}$ is put into a softmax function to classify and produce the output of the fully connected layer. We consider $l \in \{1, ..., L\}$ as the classification group number, $\hat{Y}_l \in \hat{\boldsymbol{Y}}$ as the output prediction,

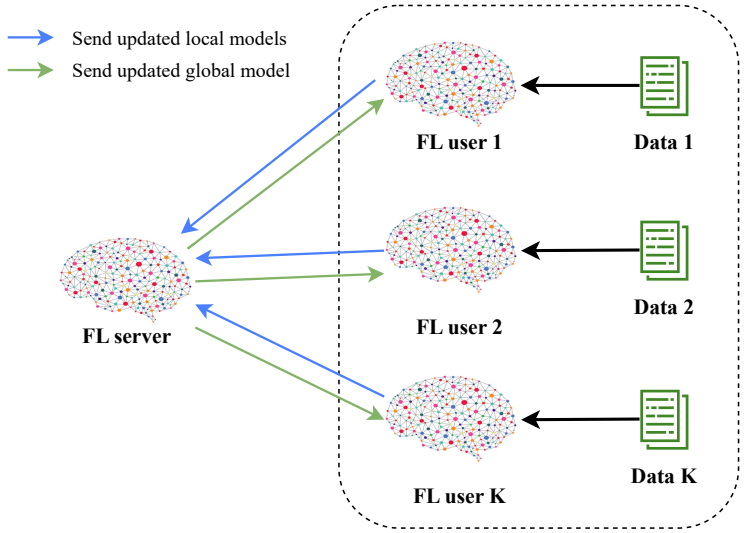


Figure 1.5: Illustration of an FL process.

the probability that an output prediction $\hat{Y}$ belongs to group l can be calculated as follows:

$$\begin{aligned} p(\hat{Y}_l = l|\boldsymbol{I}_{e,i}, \boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i}) &= softmax(\boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i}) \\ &= \frac{e^{\boldsymbol{W}_{e,i}\boldsymbol{I}_{e,i}+\boldsymbol{b}_{e,i}}}{\sum_l e^{\boldsymbol{W}_{e,l,i}\boldsymbol{I}_{e,i}+\boldsymbol{b}_{e,l,i}}}, \end{aligned} \tag{1.17}$$

where $\boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i}$ are the weights and biases of the fully connected layer at iteration i, respectively; and $\boldsymbol{W}_{e,l,i}, \boldsymbol{b}_{e,l,i}$ as weights and biases of the fully connected layer at iteration i to classify an output prediction into class l. Based on equation (1.17), we can calculate a vector of prediction $\hat{\boldsymbol{Y}}$ which includes output prediction $\hat{Y}_l$ belonging group l with probability p as follows:

$$\hat{\boldsymbol{Y}} = \underset{l}{\text{argmax}}[p(\hat{Y}_l = l|\boldsymbol{I}_{e,i}, \boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i})]. \tag{1.18}$$

1.2.2 Collaborative Learning

1.2.2.1 Federated Learning

Federated Learning (FL) is an emerging distributed learning framework that allows multiple devices to distributedly train a shared model without the need of collecting and aggregating all data from all the nodes in the network, thereby protecting data privacy and reducing communication overhead. Typically, an FL system consists of an FL server and N FL users. Each user $i \in \mathcal{N}$, $\mathcal{N} = \{1, \ldots, N\}$, in the system possesses a private dataset D_i and a local model $\mathbf{w}_i$, whereas the FL server has a global model $\mathbf{w}_G$. As illustrated in Figure 1.5, at the beginning of an FL process, the FL server broadcasts the initial global model $\mathbf{w}_G^1$ to every user in the network. Then, each user i trains the received model locally using its private dataset D_i to generate the gradient $\mathbf{g}_i^1$ and the updated local model $\mathbf{w}_i^1$ [24]:

$$\mathbf{w}_i^1 \leftarrow \mathbf{w}_G^1 - \mu \mathbf{g}_i^1. \tag{1.19}$$

After the training process finishes, the users send their updated local models $\mathbf{w}_i^1 (\forall i \in \mathcal{N})$ to the server. The server then aggregates all the updated weights of local models into the global model using an aggregation algorithm, e.g., the FedAvg Algorithm [24]:

$$\mathbf{w}_G^2 \leftarrow \sum_{i=1}^{T} \frac{t_i}{t} \mathbf{w}_i^1. \tag{1.20}$$

In equation (1.20), t is the total number of samples, T is the number of local models, and t_i is the number of samples of user i. The aggregated global model weight $\mathbf{w}_G^2$ is then sent to the users for training in the next iteration. The FL process is repeated until a termination criterion is met, e.g., the global loss function converges. As a result, each user can receive updated learning knowledge from others without sharing their own private data. However, the conventional FL approaches

require the dataset of each user to have the same characteristics such as features or labels. Therefore, they might not be effective in highly dynamic and decentralized networks such as the cyberattack detection for IoT networks [14, 25].

1.2.2.2 Transfer Learning

Conventional deep learning uses a dataset collected from the network to learn and predict the network behaviors. However, when the dataset is small and lack of useful information, it will dramatically affect the accuracy of the trained model [26]. Transfer Learning (TL) is a technique that allows a model fed by a small dataset to improve its learning by inheriting the knowledge learned from other models with much better data quality. This method can also overcome the limitations of conventional FL by allowing datasets with different characteristics (e.g., features or labels) to share and exchange their knowledge. As a result, this technique has a wide range of applications in practice [27, 28].

TL is generally defined based on two fundamental concepts, i.e., domain and task. A domain $\mathcal{D}$ consists of a feature space $\mathcal{K}$ and a marginal probability distribution $P(\mathbf{K})$ where $\mathbf{K} = \{k_1, ..., k_n\} \in \mathcal{K}$ and n is the number of features in vector $\mathbf{K}$ i.e., $\mathcal{D} = \{\mathcal{K}, P(\mathbf{K})\}$. Given $\mathcal{D}$, a task is defined by a function $\Omega(\cdot)$ that aims to map domain $\mathcal{D}$ to a label space $\mathcal{L}$ of labels $\mathbf{Y}$. We denote by $\mathcal{D}_\mathrm{S}$, $\mathcal{T}_\mathrm{S}$, $\mathcal{D}_\mathrm{T}$, and $\mathcal{T}_\mathrm{T}$ the source domain, source task, target domain, and target task, respectively. The main objective of TL is to utilize the knowledge from the source, i.e., $\mathcal{D}_\mathrm{S}$ and $\mathcal{T}_\mathrm{S}$, to improve the learning process of the target task. Based on the availability of labeled data and the difference among $\mathcal{D}_\mathrm{S}$, $\mathcal{T}_\mathrm{S}$, $\mathcal{D}_\mathrm{T}$, and $\mathcal{T}_\mathrm{T}$, TL can be classified into inductive ($\mathcal{D}_\mathrm{S} = \mathcal{D}_\mathrm{T}$, $\mathcal{T}_\mathrm{S} \neq \mathcal{T}_\mathrm{T}$, labeled target data), transductive ($\mathcal{D}_\mathrm{S} \neq \mathcal{D}_\mathrm{T}$, $\mathcal{T}_\mathrm{S} = \mathcal{T}_\mathrm{T}$, labeled source data), and unsupervised TL (unlabeled target and source data) [27–29]. With the ability to inherit knowledge from bigger sources, TL can be applied to address numerous practical problems, especially in cybersecurity to

detect cyberattacks in IoT networks [17, 28, 30].

1.3 Literature Review

In this section, we first present the advantages and the limitations of existing work on using various DL models for cyberattack detection. We then highlight the main contributions of this thesis.

1.3.1 Machine Learning for Cyberattack Detection in IoT Networks

In this section, we review several emerging machine learning techniques that have been used to detect cyberattacks in IoT networks.

1.3.1.1 Deep Learning for Cyberattack Detection in IoT Networks

Recently, with outstanding classification ability, Machine Learning (ML) techniques, especially deep learning (DL), have been widely applied to cyberattack detection problems. Particularly, DL models can effectively learn the signatures of various cyberattack types. Moreover, DL models even can detect new types of attacks that have never been learned/trained before [11]. There is rich literature proposing DL approaches for cyberattack detection. In [31], a deep neural network (DNN) model is developed to detect zero-day attacks based on two types of data, i.e., network activities and local system activities. The results show that for most of the datasets, the proposed DNN can achieve a higher detection accuracy and lower false-positive rate compared to those of the other conventional machine learning classifiers such as K-Nearest Neighbors (KNN), and Support Vector Machine (SVM). In [32], the authors build IoT systems and perform cyberattack experiments on them to collect a dataset named N-BaIoT dataset. In these experiments, the authors deploy DoS and DDoS attacks using BASHLITE and Mirai on 9 IoT devices. After that, they capture network traffic and extract their features into a dataset. They then use

the dataset to train the autoencoder neural network to detect attacks. The simulation results show that the accuracy of detection can reach 100% in terms of True Positive Rate. However, in [33], the authors pointed out that the N-BaIoT lacks the telemetry data of IoT sensors and the data traces of operating systems. They deployed cyberattack experiments and then collected a dataset named ToN-IoT to address the problem of N-BaIoT. After that, they used DNN, Naive Bayes (NB), and many other ML methods to analyze their dataset. All of them demonstrated their performance with the Area Under the Curve (AUC) above 90%. In [34], the authors summarized IoT security threats, provided the taxonomy of ML and DL for IoT security at different levels, and analyzed their challenges and future research directions.

Nevertheless, DL-based cyberattack detection systems are also facing numerous practical challenges. Particularly, conventional DL approaches usually require a huge amount of data to achieve a high performance. However, in various applications, data are very difficult to collect because they are often stored locally on user devices such as IoT devices, smartphones, and wearable devices. This poses a threat to user privacy because sensitive data (e.g., location and private information) have to be sent over the network and stored at the centralized server for processing. Beside the privacy concerns, transmitting such a collectively large amount of data also imposes an extra communication burden over the network. Consequently, these limitations have been hindering the effectiveness of DL techniques in cyberattack detection systems.

1.3.1.2 Federated Learning for Cyberattack Detection in IoT Networks

With the advent of FL, numerous research studies have been performed to apply this framework to cyberattack detection, especially in environments with a large of devices such as IoT and mobile edge networks. In [35], an FL framework is

proposed for cyberattack detection in an edge network. In this network, the data for intrusion detection are stored locally at each edge node. The edge nodes train their data locally and send their model weights to an FL server for aggregating. After aggregation, the FL server sends the weights back to all edge nodes. In this way, each edge node can benefit from data and training of the other nodes while protecting its privacy and reducing the communication burden of the network. Experiments with the NSL-KDD datasets showed that the proposed approach could achieve an accuracy of up to 99.2%. In [36], the authors proposed DÏoT which is a self-learning distributed system for anomaly detection in IoT devices. In this paper, the authors performed experiments on 30 IoT devices and showed that their proposed model could address various new types of attacks, and their model could reach a detection rate of up to 95.6% at around 257ms. Another FL approach is proposed in [37] for anomaly detection in IoT networks. In this paper, the authors deployed a DL model that exchanged learned weights between a gated recurrent unit (GRU) and a central server. The simulation results showed that their proposed model could achieve an accuracy of up to 99.5% with the lowest number of false alarms. In [38], the authors reviewed the state of the art of FL and provided their vision of FL in IoT. They also presented the advantages of using FL in IoT services and IoT applications, especially in attack detection and data privacy. At the end of the survey, the authors also provided the research challenges and future direction in using FL in IoT systems.

1.3.1.3 Transfer Learning for Cyberattack Detection in IoT Networks

Although FL can effectively address the privacy and communication load concerns of conventional ML for cyberattack detection, they are still facing several challenges. Particularly, FL usually requires high-quality and labeled data for training. However, collecting and labeling such data is expensive and time-consuming,

especially for large-scale systems. On the other hand, unlabeled data are often abundant in environments such as IoT and mobile edge networks. Thus, a deep TL technique was proposed for IoT intrusion detection in [17] based on network activities, utilizing both labeled and unlabeled data. In this approach, the authors employed two AEs. The first AE was trained with labeled data, while the second AE was trained with unlabeled data. Then, the knowledge was transferred from the first AE to the second AE by minimizing the Maximum Mean Discrepancy (MMD) distances between their weights. Experiments over nine IoT datasets were conducted to show that the proposed approach could achieve higher Area Under the Curve (AUC) scores compared to those of many other approaches. In [39], the authors proposed a dependable intrusion detection system model based on TL techniques to detect abnormal behaviour of IoT systems. In this paper, the authors designed a TL model that used the knowledge of the ResNet model to detect attacks in a small amount of data. The simulation results showed that their proposed model which could achieve an F1-Score of up to 86% outperformed other state-of-the-art models such as CNN, RNN, and LSTM. In addition, in [40], the authors studied the use of TL for intrusion detection in IoT devices. In this paper, the authors explored TL to transfer learned knowledge for both generating appropriate intrusion algorithms for new IoT devices and detecting new types of attacks. The simulation results showed that their proposed TL model could achieve a high accuracy in detecting attacks with a low energy.

Based on the aforementioned observations, in Chapter 2, we will propose a collaborative learning framework that can leverage the strengths of both FL and TL to address the limitations of ML-based intrusion detection systems, e.g., lack of labeled data, privacy, and heterogeneous data feature space. Moreover, in our approach, each IoT network has a separate model that is fine-tuned specifically for that network, therefore the model is more effective for that network cyberattack detection

compared to FL frameworks with a single model for all networks. Furthermore, our proposed system model can utilize knowledge from both source and target data in the network instead of only transferring knowledge from a single source as proposed in most of the mentioned TL frameworks [17,41–43], thereby mitigating the negative transfer problem.

1.3.2 Machine Learning for Attack Detection in Blockchain Systems

This section presents ML techniques that have been proposed to detect cyber-attacks in blockchain systems.

1.3.2.1 Machine Learning for Attack Detection in Network Traffic

ML has been considered as a highly-effective solution to detect cyberattacks for blockchain networks [44]. In particular, in [45], the authors proposed to use Random Forest and XGBoost to detect attacks in a blockchain-based IoT system. The results showed that this solution could identify different types of attacks and normal behaviors with an accuracy of up to 99%. However, they only test their results on the BoT-IoT dataset that is not real blockchain traffic. Similarly, in [46], the authors proposed an ML-based method, called bidirectional long short-term memory (BiLSTM) to detect attacks in an IoT network before the data is stored in the blockchain network. Although the results also showed that they could detect different kinds of attacks with an accuracy of up to 99%, they were validated only on conventional network datasets such as UNSW-NB15 and BoT-IoT datasets. These datasets were collected in conventional computer networks and thus cannot reflect actual traffic in blockchain networks. In particular, these datasets have just general attacks in computer networks without specific attacks in blockchains, e.g., changes in blockchain transactions, incorrect consensus protocol, or the break of the chain of blocks.

To the best of our knowledge, there are a few studies that consider using the real blockchain traffic, e.g., try to generate artificial data or try to create data to simulate an attack for blockchain networks to train ML models such as [47–49]. Specifically, in [47] the authors proposed a method to collect blockchain traffic data. First, they captured traffic samples from a public Bitcoin node and used them as normal network data. Then, for the malicious traffic data, the authors performed DoS and Eclipse attacks on a target device (this device was created to become a node in the Bitcoin network). After that, the collected data was used to train an autoencoder deep learning model. This solution showed an accuracy of attack detection up to 99%. In [48], the authors used a public dataset and a private dataset from their testbed. Then, they proposed to use a Long Short-Term Memory Network (LSTM) to learn the properties of normal samples in the datasets. After that, they deployed a Condition Generative Adversarial Networks (CGAN) model to generate the artificial Low-rate Distributed DoS (LDDoS) attack samples for their blockchain dataset. The results showed the accuracy of classification up to 93%. In addition, in [49], the authors performed a DDoS attack namely Link Flood Attack (LFA) on a simulation Ethereum network and collected the traceroute records of the network in both normal and attack behaviors. After that, the authors used the Recurrent Neural Network (RNN) to analyze the traceroute records to identify the attacks in the network. The results showed that the attack detection rate could achieve nearly 99%. In [50], the authors developed a framework based on blockchain to detect only one specific attack, i.e., replay attacks, for a power system. Moreover, in [51], the authors proposed using blockchain and the Support Vector Machine (SVM) to detect cyberattacks for multimicrogrid systems. Differently, in our work, we aim to develop a decentralized learning model that can detect different types of attacks (i.e., Denial of Service, Password Brute-Force, and Flooding of Transactions) for blockchain-based systems.

From all the above works and others in the literature, we can observe two main challenges for ML-based intrusion detection systems in blockchain networks which have still not been addressed. In particular, the first challenge is the lack of synthetic data from laboratories for training ML models. Most of the current works, e.g., [45] and [46] are using conventional cybersecurity datasets (e.g., UNSW-NB15 and BoT-IoT) to train data. However, these datasets are not designed for blockchain networks, thus they are not appropriate to use in intrusion detection systems in blockchain networks. Other works, e.g., [47–49], try to build their own datasets for blockchain networks, e.g., by obtaining the normal samples from the Bitcoin network [47], creating simulation experiment to detect the LFA [49] and generating artificial attack samples by CGAN [48]. However, these methods have several issues. First, normal samples of transactions from the Bitcoin network may include attacks from public blockchain network, but all collected data are classified and labeled to be normal data. Second, the simulation experiment in [49] is to generate traceroute records only for the LFA so they cannot extend to other attacks. Furthermore, it is difficult to evaluate the effects of artificial attack samples in [48] whether they can simulate a real attack on a blockchain network or not. Another challenge we can observe here is that all of the current ML-based intrusion detection solutions for blockchain networks are based on centralized learning models, i.e., all data is collected at a centralized node for training and detection. However, this solution is not suitable to deploy in blockchains as they are decentralized networks. Specifically, nodes in blockchain networks may have different data to train, and due to privacy concerns, they may not want to share their raw data* with a centralized node (or other nodes) for training processes. Moreover, sending a huge amount of data to the network will not only cause excessive network traffic but also risk compromising the data

*The raw data means the network traffic data of a local network that can be classified into normal or attack data, that will be used for the learning process. This data usually contains sensitive information (e.g., cryptographic keys, usage ports, or local network bandwidth) that the node does not want to share with other nodes in the network.

integrity of blockchain networks.

1.3.2.2 Federated Learning for Attack Detection in Transactions and Smart Contracts

There are several works trying to deal with attacks on transactions and Smart Contracts (SCs) in blockchain networks. In [52], the authors proposed to convert the source codes of smart contracts into vectors. They then used bidirectional long-short-term memory to identify abnormal patterns of vectors to detect re-entrancy attacks. The simulation results showed that their proposed model could achieve 88.26% F1-Score and 88.47% accuracy in detecting re-entrancy attacks. In [53], the authors proposed to use feature extraction to analyze the Bytecode of SCs. This approach is motivated by the fact that the characteristics of attacks are often expressed as sets of hexadecimal numbers embedded inside bytecodes. In this paper, the authors used various types of ML models to detect 6 types of attacks with an F1-score of up to 97%. Even though the methods in [52, 53] can detect various types of attacks, they need to use source codes of SCs in high-level programming languages (e.g., Solidity). It is worth noting that when an SC is created, the SC creates corresponding transactions for execution and then sends them to ENs for the mining process. From the EN point of view, we only can observe transactions with the encoded content (e.g., Bytecode) in their features. In real-time attack detection, we need to analyze this content to find out the insight attacks in transactions and SCs.

Unlike the above DL approaches, in [54], the authors also studied the Bytecode. They proposed to use the attack vector method to directly analyze the Bytecode. This approach could effectively detect various specific attacks by using pre-defined vectors. Hence, this method is difficult to extend to various types of new attacks. In addition, even though the attack detection ability could achieve up to 100% in var-

ious types of attacks (e.g., re-entrancy, delegatecall, overflow, etc), the authors only tested this method in a small scale of data (about 100 samples). In [55] the authors proposed to use Graph embedding to analyze Bytecode. To do this, the authors converted the Bytecode of SC into vectors and then compared the similarities between the vectors of SC to detect the insight attacks of SC. The experimental results showed that this method could achieve a precision of up to 91.95% in detecting attacks. Both [54] and [55] have to use source code to analyze the bytecode. In [56], the authors introduced a smart contract security testing approach with the aim of identifying the suspicious behaviors associated with vulnerabilities of smart contracts in blockchain networks. According to their evaluations, the proposed framework completely rejected about 3.5% of transactions due to being untestable. Therefore, they pointed out that further Bytecode analysis could reduce this portion. In addition, in [57], the authors proposed DefectChecker which was a framework using symbolic execution to analyze Bytecode without the need for source codes. This framework could detect eight types of attacks in SCs and get an F1-score of 88%. Unlike all the above works and others in the literature, in this thesis, we introduce an innovative ML-based framework to analyze Bytecode directly from transactions without the need for source code. To do this, we propose to convert the encoded information of transactions into images. Our proposed framework can analyze these images to detect various types of attacks in both transactions and SCs. In this way, our proposed framework is flexible and can detect different types of attacks easier. Moreover, all of the methods above focus on centralized learning. To implement those methods, all the data needs to be gathered in a centralized server for learning and analysis. However, blockchain is a decentralized environment and Ethereum nodes are distributed worldwide. Thus, gathering all blockchain data to perform training and testing is impractical and ineffective.

To the best of our knowledge, at the time we perform this work, there is still

no work on FL to deal with the aforementioned challenges in blockchain systems. Therefore, in Chapter 3, we propose a collaborative learning model to analyze the network traffic for detecting cyberattacks in blockchain networks. In addition, in Chapter 4, we propose a collaborative learning model that can analyze directly the bytecode to detect insight attacks in smart contracts and transactions.

1.4 Thesis Organization and Contributions

In a decentralized network, it is difficult to gather local data from various nodes to feed the centralized deep learning-based cyberattack model. Thus, it is important to develop learning models that can efficiently work in decentralized networks, e.g., IoT and blockchain without the need of gathering all local data but still provide high accuracy like centralized models. The detailed contributions and the rest of this thesis are presented in the following sections.

1.4.1 Chapter 2 - Federated Transfer Learning for IoT Networks

Chapter 2 presents the proposed federated transfer learning model for cyberattack detection. In particular, Section 2.1 first describes the system model and our proposed approach. Section 2.2 discusses the datasets, experiment setup, and experimental results. Finally, the conclusion and future work are given in Section 2.3. The major contributions of this chapter regarding detecting cyberattacks for an unlabelled IoT network are summarized as follows:

- We propose a novel collaborative learning framework that can effectively detect cyberattacks in decentralized IoT systems. By combining the strengths of FL and TL, our proposed framework can improve learning efficiency and the accuracy of cyberattack detection in comparison with the conventional DL-based cyberattack detection systems.

- We propose an effective TL approach that can allow the DL model from a rich-data network to transfer useful knowledge to a low-data network even they have different features for cyberattack detection in IoT networks.

- We perform extensive experiments on recent real-world datasets including N-BaIoT, KDD, NSL-KDD, and UNSW to evaluate the performance of the proposed collaborative learning framework. The results show that our proposed approach can achieve an accuracy of up to 99% and an improvement of up to 40% over the unsupervised learning approach.

1.4.2 Chapter 3 - Federated Learning for Attack Detection in Blockchain Networks

Chapter 3 presents our collaborative learning model for cyberattack detection in blockchain networks. Specifically, Section 3.1 describes the blockchain fundamental and our system model. Section 3.2 provides details about our proposed model. Section 3.3 describes our experiment setup, dataset collection, and evaluation method. The simulation and real-time experimental results are discussed in Section 3.4. Finally, conclusions and future works are given in Section 3.5. The main contributions of this chapter regarding detecting cyberattacks for blockchain networks can be summarized as follows:

- We set up experiments in our laboratory to build a private blockchain network with the aims of not only obtaining real blockchain datasets but also testing our proposed learning model in a real-time manner. To the best of our knowledge, this is the first dataset obtained from a laboratory for studying cyberattacks in blockchain networks, and thus we expect that our proposed BNaT dataset can promote the development of ML-based intrusion detection solutions in blockchain networks in the near future. More details of the dataset can be

found at the link [†].

- We build an effective tool named Blockchain Intrusion Detection (BC-ID) to collect data in the blockchain network. This tool can extract features from the collected network traffic data, filter attack samples in network traffic, and exactly label them in a real-time manner.

- We propose a collaborative decentralized learning model to not only improve the accuracy of identifying attacks but also effectively deploy in decentralized blockchain networks. This model enables fullnodes in the blockchain network to effectively share their trained models to improve cyberattack detection efficiency without the need of sharing their raw data.

- We perform both intensive simulations and real-time experiments to evaluate our proposed framework. Both simulation and experimental results clearly show the outperformance of our proposed framework compared with other baseline ML methods. Furthermore, our results reveal various important information in designing and implementing learning models in blockchain networks in practice, e.g., real-time monitoring and detecting attacks.

1.4.3 Chapter 4 - Federated Learning for Attack Detection in Transactions and Smart Contracts

Chapter 4 introduces our proposed collaborative learning to detect cyberattacks in blockchain transactions and smart contracts. In particular, Section 4.1 first provides the overview of our model. After that, Section 4.2 describes in detail our proposed model for detecting attacks in transactions and smart contracts. Section 4.3 then discusses the simulation and real-time experimental results of this work. Finally, conclusions are drawn in Section 4.4. The main contributions of this chapter

[†]https://avitech-vnu.github.io/BNaT

regarding detecting attacks for transactions and smart contracts can be summarized as follows:

- We implement a blockchain system and perform experiments to collect the Attacks on Blockchain Transactions Dataset (ABTD) dataset. To the best of our knowledge, this is the first dataset with cyberattacks on transactions and SCs of a blockchain system that is synthesized in a laboratory.

- We develop a Blockchain Code Extraction and Conversion Tool (BCEC) that can collect transactions, extract their features, and convert them into images to build a dataset. This tool can be implemented in real-time to support the analysis of the attack detection framework.

- We develop a real-time attack detection framework that can be deployed at the Ethereum nodes to detect attacks in transactions and SCs for a blockchain network. In our framework, the Ethereum nodes can detect attacks in transactions and SCs in real-time at about 2,150 transactions per second.

- We propose a collaborative learning framework that can efficiently detect attacks in a blockchain network. In our framework, each Ethereum node can exchange learning knowledge with others and then aggregate a new global model without any centralized model. In this way, our framework can achieve high accuracy at about 94% without exposing the local dataset of Ethereum nodes over the network.

- We perform both simulations and real-time experiments to evaluate our proposed framework. Our proposed framework can achieve accuracy up to 94% in simulation and 91% in real-time experimental results. In addition, our framework has the capacity to analyze various types of transaction features, expanding the detection capabilities for the diversity of attacks.

1.4.4 Conclusions and Potential Research Directions

Chapter 5 includes the conclusion and highlights various potential future research directions of this thesis.

Chapter 2

Collaborative Learning for Cyberattack Detection Systems in IoT Networks

In this chapter, we propose a novel collaborative learning framework that utilizes the strengths of both TL and FL to address the limitations of conventional DL-based cyberattack detection systems. Particularly, we consider a scenario with two different IoT networks*. The first network (source network) has an abundant labeled data resource, while the second network (source network) has very little data resource (and most of them are unlabeled). The IoT network can be used for many purposes such as smart farming, manufacturing, smart city, and smart home. Each IoT network usually includes an IoT gateway and various IoT devices. The IoT gateway serves as a "gate" to control and monitor all traffic in and out of the IoT network. Here, unlike most of the current works that assume the data in these networks has the same features [58], we consider a much more practical and general case in which data at these two networks may have different features. To address the problem of dissimilar feature spaces of the target and source networks, we propose to transform them into a new joint feature-space. In this case, at each learning round of the FL process, trained models of target and source networks can be exchanged through the joint feature-space. Thus, by periodically exchanging and updating the trained model, the target network can eventually achieve the converged trained deep neural network that can predict attacks with high accuracy (thanks to useful "knowledge" transferred from the source network). Besides exchanging and updat-

*The cases with multiple networks can be straightforwardly extended, e.g., by scheduling for networks to exchange information in order.

ing the learning model iteratively, we use a small number of mutual samples between two networks to mitigate the negative transfer learning. More importantly, unlike FL where networks try to train a joint global model, our proposed framework enables the participating networks to obtain their particular trained models that are specific to their networks, i.e., better predict attacks for particular networks with different data structures. Extensive experiments on recent real-world datasets, including N-BaIoT [32, 59], KDD [60], NSL-KDD [61] and UNSW [62] show that our proposed framework can achieve an accuracy of up to 99% and an improvement of up to 40% over the unsupervised learning approach.

The rest of this chapter is organized as follows. Section 2.1 first describes our federated transfer learning model for cyberattack detection. Section 2.2 then discusses the datasets, experiment setup, and experimental results. Finally, the conclusion and future work are given in Section 2.3.

2.1 Proposed Federated Transfer Learning Framework for Cyberattack Detection in IoT Networks

2.1.1 System Model

The conventional FL model requires to use a centralized server to maintain and aggregate all the trained models in the whole learning process. However, this may lead to a high cost to maintain and may not be effective to deploy in IoT networks. Thus, in this work, we propose a federated transfer learning model that allows the learning process to be performed more flexibly and effectively in IoT environments. In particular, we consider a network which has unlabeled data (e.g., Network B as illustrated in Figure 2.1), and it wants to learn more knowledge from other networks with abundant labeled data. In this case, this network will connect with a target network (e.g., Network A as illustrated in Figure 2.1) and nominate itself as a cen-

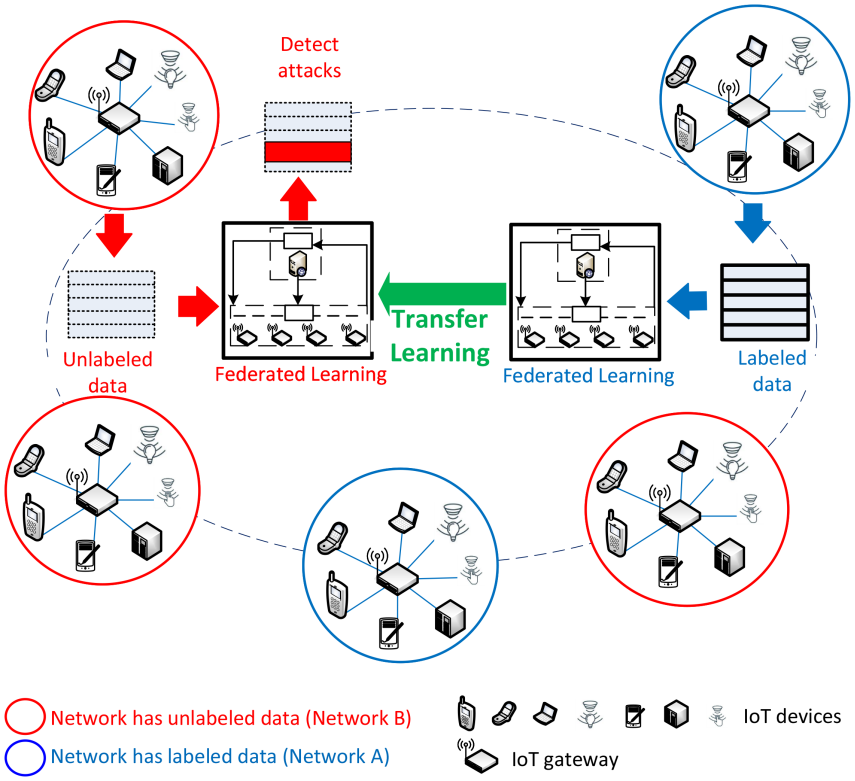


Figure 2.1: Illustration of a system model for cyberattack detection in IoT networks.

tralized node which can train its data as well as perform TL to exchange knowledge with the target network.

As presented in Table 2.1, we denote a labeled cybersecurity dataset $\mathbf{D}^A = \{\mathbf{X}^A, \mathbf{Y}^A, F^A\}$ of Network A with $(\mathbf{X}^A, \mathbf{Y}^A) = \{\mathbf{x}_1^A, y_1^A, \mathbf{x}_2^A, y_2^A, \ldots, \mathbf{x}_{M_A}^A, y_{M_A}^A\}$ where M_A is the number of samples of dataset $\mathbf{D}^A$, $\mathbf{x}_i^A, i \in \{1, ..., M_A\}$, is a vector of a sample data and $y_i^A, i \in \{1, ..., M_A\}$, is a corresponding label in dataset $\mathbf{D}^A$. In contrast, Network B has an unlabeled cybersecurity dataset $\mathbf{D}^B = \{\mathbf{X}^B, F^B\}$ with $\mathbf{X}^B = \{\mathbf{x}_1^B, \mathbf{x}_2^B, \ldots, \mathbf{x}_{M_B}^B\}$ where M_B is the number of samples of dataset $\mathbf{D}^B$, $\mathbf{x}_i^B, i \in \{1, ..., M_B\}$, is a vector of a sample data in dataset $\mathbf{D}^B$. F^A and F^B are the numbers of features of datasets in Network A and Network B, respectively. The proposed model will perform TL between two neural networks by minimizing the total loss J to predict the output of the system $P(\mathbf{z}_i^B)$ with $\mathbf{z}_i^B \in \mathbf{Z}^B, i \in \{1, ..., M_B\}$, as the output for the unlabeled dataset of Network B. In this way, the network

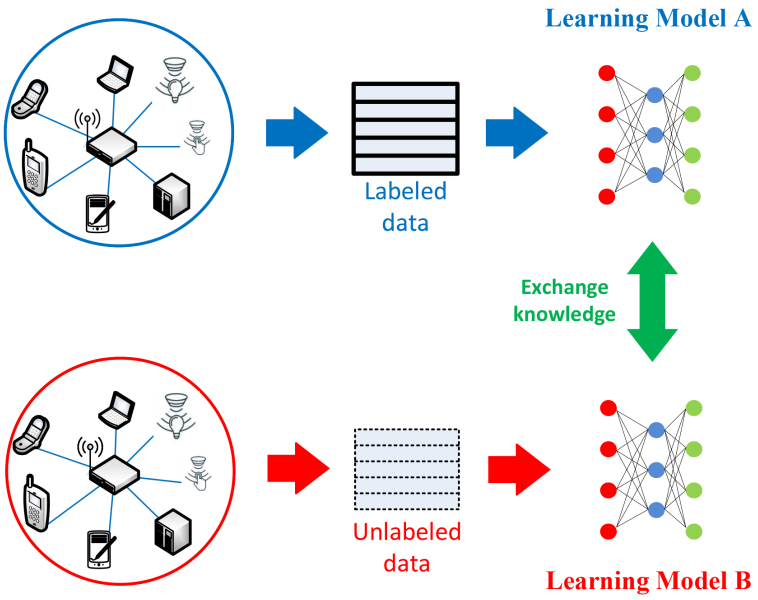
Table 2.1: Table of Notations.

Notation	Description
$\mathbf{X}^A, \mathbf{X}^B$	The matrix of samples of datasets in Network A and Network B, respectively.
$\mathbf{Y}^A, \mathbf{Y}^B$	The vector of labels of datasets in Network A and Network B, respectively.
F^A, F^B	The numbers of features of datasets of Network A and Network B, respectively.
$\mathbf{D}^A, \mathbf{D}^B$	Datasets of Network A and Network B, respectively.
M_A, M_B	The number of samples of datasets in Network A and Network B, respectively.
M_C	The number of predicted labels.
M_{AB}	The number of overlapping samples between dataset A and dataset B.
$\mathbf{W}^A, \mathbf{W}^B$	The parameter matrices of the neural networks in Network A and Network B, respectively.
$\mathbf{Z}^A, \mathbf{Z}^B$	The output matrices of the neural networks in Network A and Network B, respectively.
$\mathbf{z}_i^A, \mathbf{z}_i^B$	The output vector of an input sample data of the neural networks in Network A and Network B, respectively.
J^B	The result of the loss function in Network B.
J^{AB}	The result of the alignment loss function between Network A and Network B.
J	The result of final loss function.
w_l^A, w_l^B	The training parameters for layer-l of the neural networks in Network A and Network B, respectively.

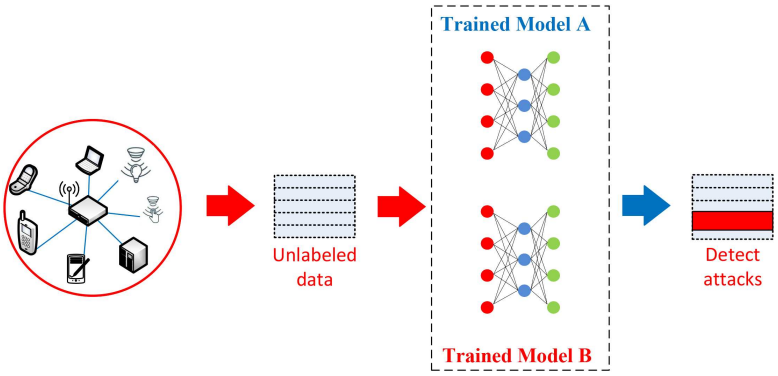
can help to improve the accuracy in identifying network traffic by learning useful knowledge from other labeled networks. Each network can be managed by an IoT gateway and possesses its private dataset. The IoT gateway uses its DL model to detect normal and abnormal traffic. It is important to note that, unlike conventional FL approaches [18], in this work, we consider a practical scenario in which the datasets of networks may have different features.

2.1.2 Proposed Federated Transfer Learning Approach for Cyberattack Detection

In this section, we propose a highly-effective federated transfer learning model



(a) FTL training process.



(b) FTL predicting process.

Figure 2.2: The FTL processes.

that can exchange knowledge between an unlabeled network and multiple networks which may have different features. To better analyze the impact of our proposed approach, we consider a specific scenario in which one labeled network is used as a source network to support an unlabeled network (i.e., target network). The scenario with one unlabeled network and multiple labeled networks can be straightforwardly extended, and we leave it for future study. Figure 2.2 describes the training and predicting processes of FTL algorithm that we use in this case. The table of notations is presented in Table 2.1. As described in the previous section, Network A

Algorithm 2.1 Federated Transfer Learning Algorithm: Training Process

1: $iteration = 0$
2: **while** $iteration \leq T$ **do**
3: Network A performs:
4: $\mathbf{z}_i^A = \mathbf{W}^A \, \mathbf{x}_i^A$.
5: Send $\{\mathbf{z}_i^A, y_i^A\}$ to Network B.
6: Network B performs:
7: $\mathbf{z}_i^B = \mathbf{W}^B \, \mathbf{x}_i^B$.
8: Send $\mathbf{z}_i^B$ to Network A.
9: Network B performs:
10: Compute $\frac{\partial J}{\partial w_l^B}$, J^B and J^{AB}, then send them to Network A.
11: Network A performs:
12: Compute $\frac{\partial J}{\partial w_l^A}$ and J, then send them to Network B.
13: Network A performs:
14: Update $w_l^A = w_l^A - \eta \frac{\partial J}{\partial w_i^A}$.
15: Network B performs:
16: Update $w_l^B = w_l^B - \eta \frac{\partial J}{\partial w_i^B}$.
17: **if** $J_{prev} - J \leq t$ **then**
18: Send stop signal to Network B.
19: Break.
20: **else**
21: $J_{prev} = J$.
22: $iteration = iteration + 1$.
23: continue.
24: **end if**
25: **end while**

and Network B have their datasets $\mathbf{D}^A$ and $\mathbf{D}^B$, respectively. They also have their model parameter matrices called $\mathbf{W}^A$ and $\mathbf{W}^B$. The outputs of two neural networks are calculated as follows:

$$\mathbf{Z}^A = \mathbf{W}^A \, \mathbf{X}^A, \tag{2.1a}$$

$$\mathbf{Z}^B = \mathbf{W}^B \, \mathbf{X}^B. \tag{2.1b}$$

We need to find the prediction function $P(\mathbf{z}_i^B) = P(\mathbf{z}_1^A, y_1^A, \ldots, \mathbf{z}_{M_A}^A, y_{M_A}^A, \mathbf{z}_i^B)$ to predict the output of the neural network of Network B. $P(\cdot)$ can be a softmax regression function to classify the data samples based on probability. To find a

high-quality predict function, we first need to minimize the loss function using the labeled dataset as follows [14]:

$$\underset{\mathbf{W}^A,\mathbf{W}^B}{\operatorname{argmin}} J^B = \sum_{i}^{M_c} j^B(y_i^A, P(\mathbf{z}_i^B)), \tag{2.2}$$

where M_c is the number of predicted labels, J^B represents the results of the loss function and $j^B(\cdot)$ is a loss function of Network B which depends on the type of output or function, i.e., the logistic loss function as in equation (1.2) with the predicted value $P(\mathbf{z}_i^B)$ and the labeled y_i^A:

$$j^B(P(\mathbf{z}_i^B), y_i^A) = \log\left(1 + \exp(-P(\mathbf{z}_i^B)\ y_i^A)\right). \tag{2.3}$$

In addition, the datasets of Network A and Network B may have various overlapping samples, and thus we can use these samples to optimize the loss function. We denote by M_{AB} the overlapping samples between the datasets of Network A and Network B. We need to minimize the alignment loss function between them as follows [14]:

$$\underset{\mathbf{W}^A,\mathbf{W}^B}{\operatorname{argmin}} J^{AB} = -\sum_{i}^{M_{AB}} j^{AB}(\mathbf{z}_i^A, \mathbf{z}_i^B), \tag{2.4}$$

where $j^{AB}(\cdot)$ represents the alignment loss function. This function can be represented in $||\mathbf{z}_i^A - \mathbf{z}_i^B||^2$ or $-\mathbf{z}_i^A\ (\mathbf{z}_i^B)'$ and $J_R^B = \sum_l^{L_B} ||w_l^B||^2$ in which L_A and L_B are the numbers of layers in neural networks of Network A and Network B, respectively, to find the final loss function that needs to be minimized [14]:

$$\underset{\mathbf{W}^A,\mathbf{W}^B}{\operatorname{argmin}} J = J^B + \gamma J^{AB} + \frac{\lambda}{2}(J_R^A + J_R^B), \tag{2.5}$$

where γ and λ are the weight parameters. We denote by w_l^A, w_l^B the training parameters for layer-l of the neural networks in Network A and Network B. The gradients

for updating $\mathbf{W}^A, \mathbf{W}^B$ are calculated as $\frac{\partial J}{\partial w_l^A}, \frac{\partial J}{\partial w_l^B}$, respectively, by the following formulas [14]:

$$\frac{\partial J}{\partial w_l^A} = \frac{\partial J^B}{\partial w_l^A} + \gamma \frac{\partial J^{AB}}{\partial w_l^A} + \lambda w_l^A, \tag{2.6a}$$

$$\frac{\partial J}{\partial w_l^B} = \frac{\partial J^B}{\partial w_l^B} + \gamma \frac{\partial J^{AB}}{\partial w_l^B} + \lambda w_l^B. \tag{2.6b}$$

The training process is presented in Algorithm 2.1. In this algorithm, we use input parameters to initialize the models, such as the learning rate η, the weight parameter γ, λ, the maximum iteration T, the tolerance t and Network A and Network B initialized model parameters $\mathbf{W}^A, \mathbf{W}^B$, to generate the output trained model parameters (i.e., $\mathbf{W}^A, \mathbf{W}^B$). Specifically, we first initialize $\mathbf{W}^A$ and $\mathbf{W}^B$. Next, we calculate $\mathbf{z}_i^A$ and $\mathbf{z}_i^B$ from the input samples of dataset $\mathbf{D}^A$ and dataset $\mathbf{D}^B$ as shown in equation (2.1). Then, Network A sends $\{\mathbf{z}_i^A, y_i^A\}$ to Network B to calculate J^B, and Network B sends $\mathbf{z}_i^B$ to Network A to calculate the alignment loss function J^{AB} and the gradients of J^B as shown in equations (2.2), (2.4), (2.5) and (2.6), respectively. In equation (2.4), we use M_{AB} as the number of mutual samples of two datasets. For example, the same IoT devices are attacked by the same types of cyberattacks in different networks. Each network extracts the attack data with different features, e.g., Network A uses timeslot, packet header, IP address while Network B uses MAC address, error packets, and frame header. The number of mutual samples is an important factor that strongly supports the learning process between two networks. After that, we calculate the final loss function J and the gradient as in equation (2.5) and equation (2.6). Finally, Network A and Network B update their model parameters based on the gradient and loss functions. This process continuously repeats until the system converges or reaches the maximum number of iterations to minimize the final loss function in equation (2.5).

Algorithm 2.2 Federated Transfer Learning Algorithm: Predicting Process

1: Network B performs:
2: $\mathbf{Z}^B = \mathbf{W}^B\ \mathbf{X}^B$.
3: Send $\mathbf{Z}^B$ to Network A.
4: Network A performs:
5: Compute $P(\mathbf{Z}^B)$ and send it to Network B.

When the training completes, the prediction process described in the Algorithm 2.2 is called to predict the final result of the unlabeled dataset $\mathbf{D}^B$. In this process, both Network A and Network B have their trained models. The main purpose of Algorithm 2.2 is using the input as model parameters $\mathbf{W}^A, \mathbf{W}^A$ and the matrix of samples of dataset $\mathbf{X}^B$ to generate the prediction out $P(\mathbf{Z}^B)$. Similar to the training process, the dataset $\mathbf{D}^B$ first goes through the trained model of Network B to calculate $\mathbf{Z}^B$. Then, Network B sends $\mathbf{Z}^B$ to Network A to archive the TL knowledge from the trained model of Network A. Network A predicts the results and sends them back to Network B to classify the attack and normal behaviors of the network.

2.1.3 Evaluation Methods

As mentioned in [63,64], the confusion matrix is typically used to evaluate system performance, especially for intrusion detection systems. We denote by TP, TN, FP, and FN "True Positive", "True Negative", "False Positive", and "False Negative", respectively. The Receiver Operator Characteristic (ROC) is created by plotting the TPR over FPR at different thresholds. Then, we use Area Under the Curve (AUC) to evaluate the performance of the algorithm in the following formula:

$$\xi = \int_{x=0}^{1} \mathrm{TP}(\mathrm{FP}^{-1}(x))\, dx. \tag{2.7}$$

In our experiments, we randomly select samples from the original dataset to test

the algorithm. In this scenario, the p-value is often used to evaluate the results of random tests and is given by

$$p_0 = F(\xi|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\xi} \mathrm{e}^{\frac{-(t-\mu)^2}{2\sigma^2}} \,\mathrm{d}\xi, \tag{2.8}$$

and

$$p = p_0 * 100, \tag{2.9}$$

in which μ is the mean and σ is the standard deviation. The results are calculated by the significant number with the following formula:

$$Sig = F^{-1}(p_0|\mu,\sigma) = \{\xi : F(\xi|\mu,\sigma) = p_0\}, \tag{2.10}$$

where Sig is the significant number that represents the results of 30 random runs and the confidence of this number is calculated by $conf = 1 - p_0$. In typical scenarios, a p-value is deemed confident when p_0 falls within the range of 0.01 to 0.05 (equivalent to p values around 1 and 5), indicating significant confidence levels of approximately 99% and 95%, respectively.

2.2 Performance Analysis

2.2.1 Datasets

In this experiment, we use four popular cybersecurity datasets namely the N-BaIoT [32, 59], KDD [60], NSL-KDD [61] and UNSW [62] datasets to evaluate the performance of the proposed method. The Network-based Detection of IoT Botnet Attacks (N-BaIoT) dataset [32, 59] includes the information collected in the setup network about the normal and attack situation. The attack was performed by an attack server to nine IoT devices. Besides, the network traffic, which is captured by a sniffer server, is used to extract the dataset. This dataset is characterized by 115

features for both normal and attack behaviors. In this dataset, the attack type is the Distributed Denial of Service (DDoS) which was implemented by two well-known botnets, namely Mirai and BASHLITE. The BASHLITE botnet includes 5 types of attacks, i.e., network scanning (scan), spam data sending (junk), UDP flooding (udp), TCP flooding (tcp), and the join of sending spam data and opening port to specific IP address (combo). Besides BASHLITE, the Mirai botnet also includes 5 types of attacks, i.e., scan, ACK flooding (ack), SYN flooding (syn), udp, and optimized UDP flooding (udpplain).

In addition to IoT datasets, we also want to evaluate our proposed solution on numerous classical intrusion detection datasets, i.e., KDD [60], NSL-KDD [61] and UNSW [62] datasets. The KDD dataset [60] includes various different kinds of network attacks simulated in military network environments. The KDD dataset has 41 features and it classifies attacks into 4 groups including Denial of Service (DoS), Probe, User to Root (U2R), Remote to Local (R2L). The NSL-KDD dataset [61] inherits the properties from KDD [60] dataset such as the features and types of attacks but eliminates the redundant samples in the training dataset and the duplicated samples in the testing dataset. Although both KDD and NSL-KDD datasets are well-known and used in numerous research works, they were developed long time ago. Thus, various modern attacks were not involved. Therefore, a recent dataset, i.e., UNSW dataset [62], is considered in this work. Unlike KDD and NSL-KDD, the feature space of this dataset includes 42 types and 9 kinds of attacks, namely DoS, Backdoors, Worms, Fuzzers, Analysis, Reconnaissance, Exploits, Shellcode, and Generic.

2.2.2 Experiment Setup

In this section, we carry out experiments using all the aforementioned datasets to evaluate the performance of the proposed solution. In this experiment, we denote

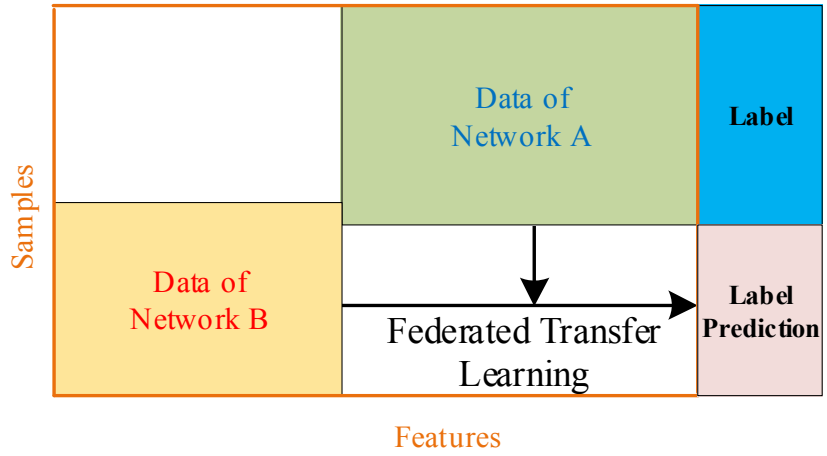


Figure 2.3: The illustration of data of participated networks used in this experiment.

by IoT1-9 the dataset names of nine IoT devices (from 1 to 9). Table 2.2 describes the total features and the representative names of datasets that we use in this experiment. Figure 2.3 also describes the separated data in each dataset in this experiment. In this experiment, the participated data are randomly selected from the dataset. Then, the selected data are separated into label data (data of Network A) and unlabeled data (data of Network B) with different features as described in Table 2.2. These data have about 10% mutual samples of total dataset samples. We experiment with two cases, i.e., the first one is with 2,000 unlabeled data and 9,577 labeled data (CASE 1), and the second one is with 10,000 unlabeled data and 47,893 labeled data (CASE 2).

In this setup, we consider a baseline solution with the state-of-the-art unsupervised deep learning (UDL) model which clusters the unlabeled data into normal and attack behaviors based on autoencoder and k-means techniques [13]. The UDL model includes an autoencoder and k-nearest neighbor to cluster the unlabeled data. In addition, we consider the second baseline solution that uses both supervised and unsupervised datasets to feed the FTL learning models. The FTL will exchange the knowledge from the supervised learning model and the unsupervised learning model

Dataset	Device name	Features of Network A	Features of Network B	Total features
IoT1	Danmini_Doorbell	85	30	115
IoT2	Ecobee_Thermostat	85	30	115
IoT3	Ennio_Doorbell	85	30	115
IoT4	Philips_B120N10_Baby_Monitor	85	30	115
IoT5	Provision_PT_737E_Security_Camera	85	30	115
IoT6	Provision_PT_838_Security_Camera	85	30	115
IoT7	Samsung_SNH_1011_N_Webcam	85	30	115
IoT8	SimpleHome_XCS7_1002_WHT_Security_Camera	85	30	115
IoT9	SimpleHome_XCS7_1003_WHT_Security_Camera	85	30	115
KDD	-	31	10	41
NSLKDD	-	31	10	41
UNSW	-	31	11	42

Table 2.2: Dataset preparation

to improve the accuracy of learning as well as increase the precise of identifying attack and normal behaviors of the unlabeled data. Then, we measure the AUC of this process 30 times to calculate the signification number of the AUC series results with both baseline solutions. Finally, we plot the reconstruction errors to analyze the convergence of the FTL algorithm for all datasets.

2.2.3 Experimental Results

In this section, we show the results of our experiments with different kinds of cybersecurity datasets.

2.2.3.1 Accuracy Comparison

In this section, we compare the performance of FTL and the UDL models in terms of the significant number of each p. Table 2.3 and Table 2.4 describe the significant number of each dataset with $p = 1, 3, 5$ corresponding to the confidence of $99\%, 97\%, 95\%$.

In general, Table 2.3 and Table 2.4 show that the significant numbers of all datasets increase as p increases. This is because in (2.10), we calculate the significant number based on a series of 30 continuous AUC results. When p increases, the AUC results increase in all tables. This demonstrates that most of the AUC results in 30

	FTL	UDL
IoT1	**85.771**	45.753
IoT2	**83.795**	63.171
IoT3	**94.286**	80.453
IoT4	**79.241**	77.885
IoT5	**90.605**	81.876
IoT6	**91.179**	82.703
IoT7	**90.670**	85.183
IoT8	**82.960**	65.256
IoT9	**83.222**	73.072
KDD	**99.315**	80.477
NSLKDD	**98.485**	83.025
UNSW	**97.072**	68.449

(a) The results with $p = 1$.

	FTL	UDL
IoT1	**87.398**	49.770
IoT2	**85.672**	65.793
IoT3	**94.896**	81.070
IoT4	**81.672**	77.885
IoT5	**91.517**	82.013
IoT6	**92.059**	82.703
IoT7	**92.030**	86.013
IoT8	**85.197**	68.161
IoT9	**85.072**	73.078
KDD	**99.395**	81.304
NSLKDD	**98.534**	83.450
UNSW	**97.141**	69.124

(b) The results with $p = 3$.

	FTL	UDL
IoT1	**88.259**	51.897
IoT2	**86.666**	67.181
IoT3	**95.220**	81.397
IoT4	**82.959**	77.885
IoT5	**92.000**	82.085
IoT6	**92.525**	82.703
IoT7	**92.750**	86.453
IoT8	**86.381**	69.700
IoT9	**86.052**	73.082
KDD	**99.438**	81.742
NSLKDD	**98.561**	83.675
UNSW	**97.177**	69.482

(c) The results with $p = 5$.

Table 2.3: The results with multiple datasets in CASE 1.

	FTL	UDL
IoT1	**90.371**	49.783
IoT2	**68.193**	62.591
IoT3	**94.525**	83.411
IoT4	**87.050**	77.725
IoT5	**86.535**	81.954
IoT6	**87.214**	82.555
IoT7	**97.662**	79.517
IoT8	**84.609**	52.702
IoT9	**90.095**	63.803
KDD	**99.535**	84.333
NSLKDD	**98.858**	81.164
UNSW	**97.049**	66.329

(a) The results with $p = 1$.

	FTL	UDL
IoT1	**91.497**	54.079
IoT2	**72.573**	65.681
IoT3	**95.073**	83.565
IoT4	**88.538**	77.781
IoT5	**88.150**	82.160
IoT6	**88.638**	82.664
IoT7	**97.928**	81.400
IoT8	**86.691**	57.318
IoT9	**90.959**	65.559
KDD	**99.562**	84.423
NSLKDD	**98.885**	81.976
UNSW	**97.121**	66.901

(b) The results with $p = 3$.

	FTL	UDL
IoT1	**92.093**	56.354
IoT2	**74.892**	67.317
IoT3	**95.363**	83.647
IoT4	**89.326**	77.811
IoT5	**89.006**	82.269
IoT6	**89.392**	82.721
IoT7	**98.069**	82.397
IoT8	**87.793**	59.763
IoT9	**91.417**	66.489
KDD	**99.576**	84.471
NSLKDD	**98.900**	82.406
UNSW	**97.159**	67.203

(c) The results with $p = 5$.

Table 2.4: The results with multiple datasets in CASE 2.

series are higher than the significant number in the case where $p = 1$.

Table 2.3(c) shows the significant numbers of participated datasets with $p = 5$ in CASE 1. In this table, the IoT1 and UNSW datasets show a significant gap of about 30% and 40% between FTL and UDL. These results show the difficulty of clustering in recognizing the groups of samples and the advantage of collaborative learning in these datasets. The other ten datasets have gaps of around 10-20% between the two methods, which demonstrate the stability of our proposed solution for any cybersecurity dataset.

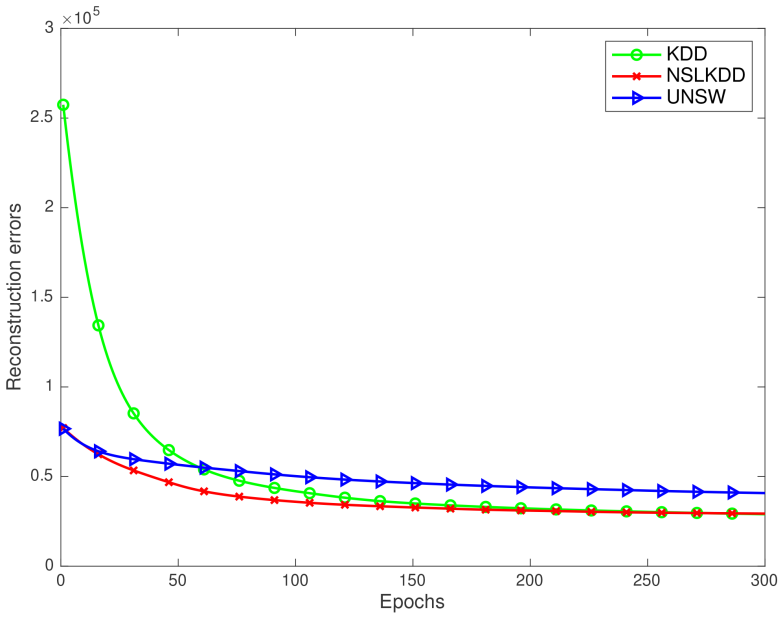
In addition, Table 2.4(c) shows the significant numbers of multiple datasets with $p = 5$ in CASE 2. In this table, the significant numbers also have a gap of around 10-40% between the two solutions. It shows the common trend that the significant numbers increase for most datasets when the number of samples increases. However, in IoT2, IoT5, and IoT6 datasets, the significant numbers slightly decrease because of the randomly selected samples from the original dataset. It also can be demonstrated by the high fluctuation of the reconstruction errors of IoT2, IoT5, and IoT6 datasets in Figure 2.5(b) compared with other datasets. However, in all studied datasets, our proposed solution still performs much better than the state-of-the-art UDL solution. These results demonstrate that our solution can work efficiently in all IoT and conventional cybersecurity datasets in detecting cyberattacks in the network.

2.2.3.2 Reconstruction Error Analysis

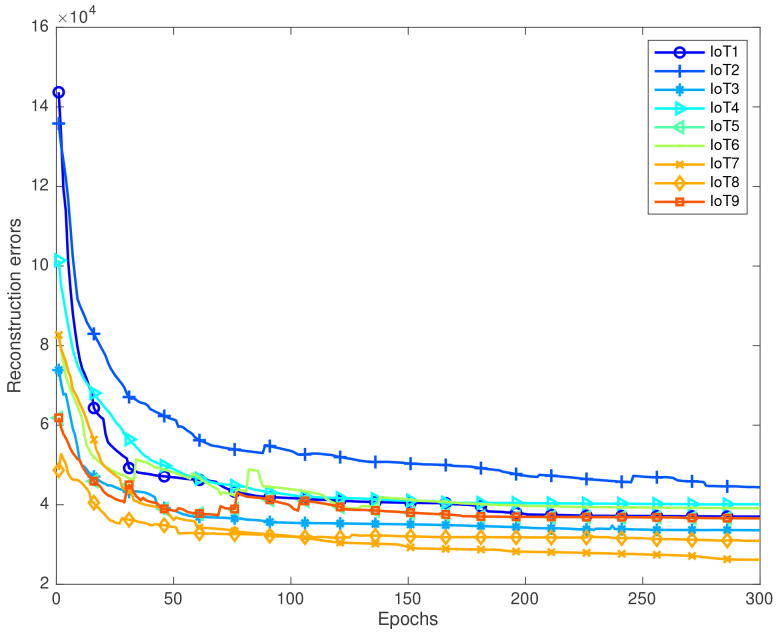
In this section, we discuss the convergence of the FTL algorithm in each dataset. Figure 2.4 describes the reconstruction errors of the nine IoT datasets and the conventional datasets like KDD, NSLKDD, and UNSW in CASE 1. Figure 2.5 describes the reconstruction errors of study datasets in CASE 2.

In Figure 2.4(a) and Figure 2.5(a), we can see that at the first few epochs, the errors are very high for KDD (up to 2.6×10^5 in CASE 1 and 12×10^5 in CASE 2), but this error dramatically reduces to 0.3×10^5 in CASE 1 and 1.5×10^5 in CASE 2 after only 200 epochs. For the NSLKDD and UNSW, they have very similar trends with 0.75×10^5 in CASE 1 and 3.8×10^5 in CASE 2 at the beginning and gradually reduce to 0.4×10^5 in CASE 1 and 1.9×10^5 in CASE 2 after 200 epochs, respectively.

Here, we observe there is a sharp reduction in reconstruction errors within the first 100 epochs of the KDD dataset, as shown in Figure 2.4(a) and Figure 2.5(a). This could be from the fact that the original KDD dataset contains a significant amount of redundant data. Many records are repeated, which means during training,

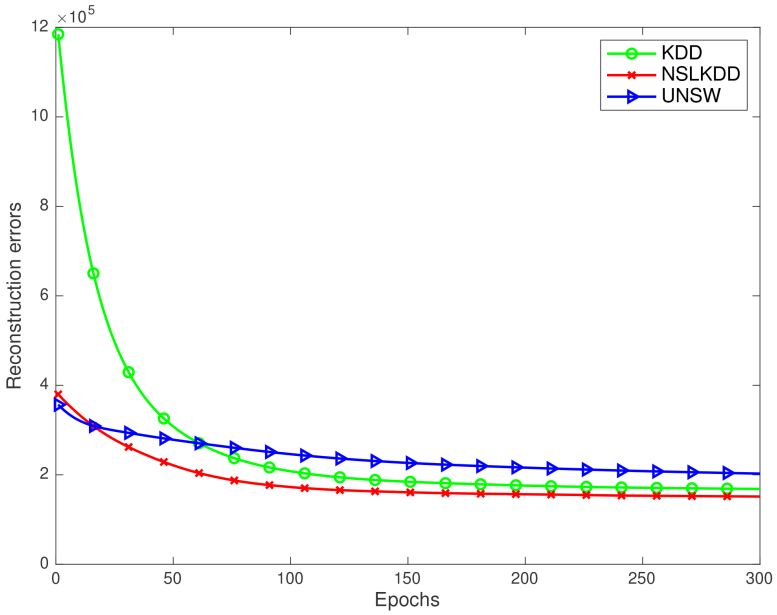


(a) The reconstruction errors of KDD, NSLKDD, and UNSW datasets.

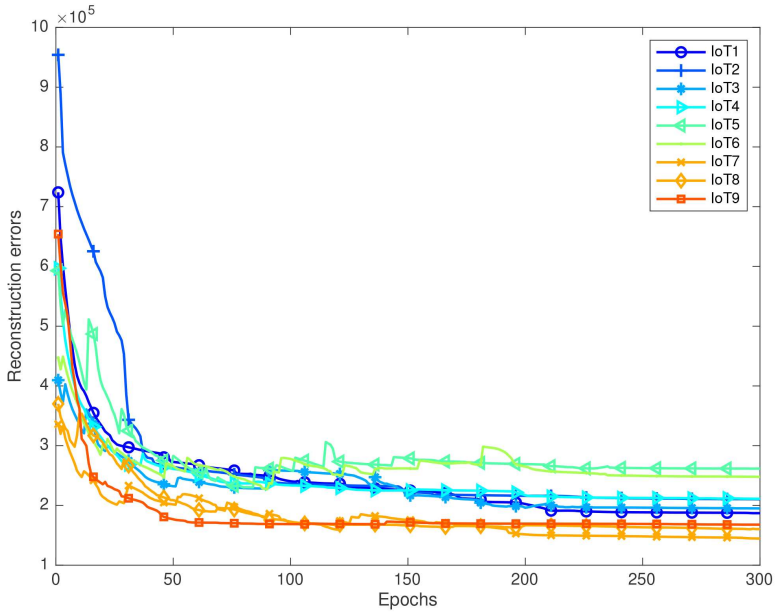


(b) The reconstruction errors of IoT datasets.

Figure 2.4: The reconstruction errors in CASE 1.



(a) The reconstruction errors of KDD, NSLKDD and UNSW datasets.



(b) The reconstruction errors of IoT datasets.

Figure 2.5: The reconstruction errors in CASE 2.

the model frequently encounters the same situations and can quickly "memorize" or fit to these records. This redundancy can artificially accelerate the convergence rate because the model is effectively learning from a less diverse set of data. After 200 epochs, the algorithm converges as all the reconstruction error curves are flattened.

Figure 2.4(b) and Figure 2.5(b) show the reconstruction errors of nine IoT datasets in both CASE 1 and CASE 2. we can observe the same trend over all datasets, i.e., all errors gradually reduce when the number of epochs increases. However, it can be observed that the trend exhibits various fluctuations in comparison with the trends in Figure 2.4(a) and Figure 2.5(a) because of the heterogeneous distribution in IoT datasets. The high fluctuation of the reconstruction errors of IoT2, IoT5, IoT6 datasets in Figure 2.5(b) also explains why their significant numbers reduce when the number of samples increases in CASE 2. However, the reconstruction errors of all studied datasets in our proposed solution dramatically decrease and become stable after 200 running epochs in both cases.

2.2.3.3 Mutual Information Analysis

As mentioned in the previous section, Network A and Network B may share a number of mutual samples. The FTL algorithm exploits the information of these mutual samples to perform the prediction for unlabeled data of Network B. This section provides the analysis results to identify how this mutual information can affect the results of label prediction. In this section, we perform the simulation in CASE 2 with a larger number of samples than in CASE 1. Figure 2.6 gives information about the variation of AUC when the percentage of mutual data increases.

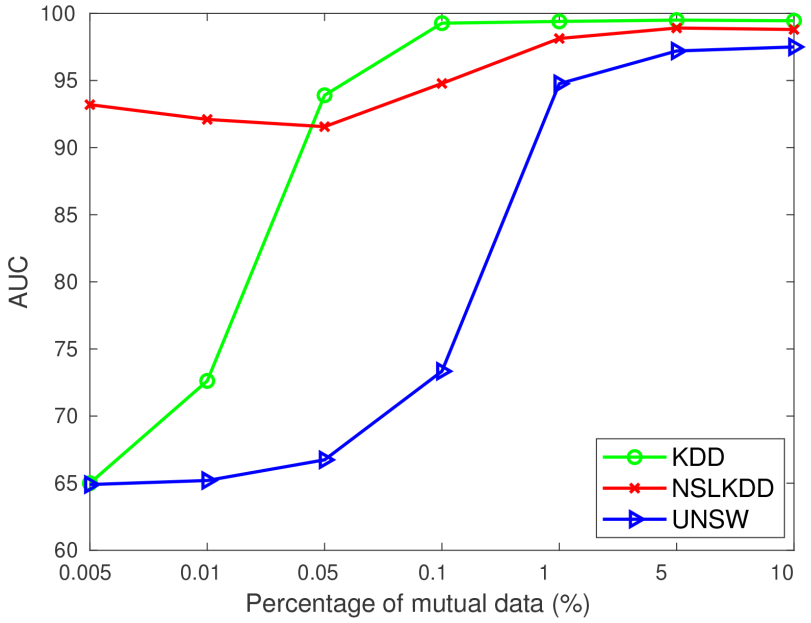
Figure 2.6(a) shows the increase of AUC on KDD, NSLKDD, and UNSW datasets when the percentage of mutual samples increases from 0.005% to 10%. The AUC of KDD and UNSW datasets sharply increase and remain stable at around 96% on the NSLKDD dataset with about 5% to 10% mutual samples. A similar trend hap-

pens with the IoT datasets in Figure 2.6(b) when the AUCs of all nine IoT datasets increase and remain stable at approximately 10% of mutual samples. From these results, it can be observed that achieving high efficiency in AUC for IoT datasets may require at least 10% of mutual data.

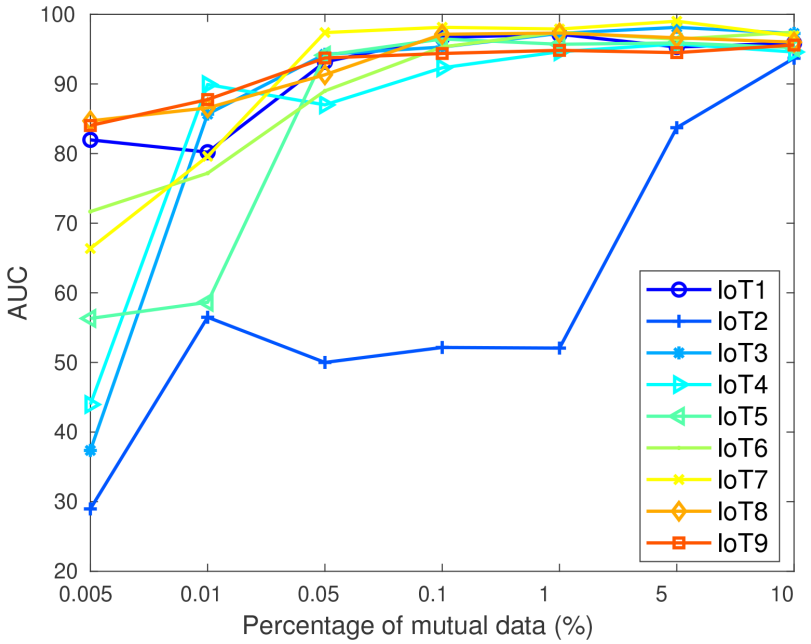
In summary, the results with 12 cybersecurity datasets show the outperformance of our proposed model in comparison with the state-of-the-art UDL in term of accuracy as shown in Table 2.3 for CASE 1 and Table 2.4 for CASE 2, especially with IoT1 and UNSW datasets. Moreover, the reconstruction errors show a fluctuation of the IoT datasets when the number of samples increases due to noise from the collected datasets of various IoT devices. Finally, we vary the amount of mutual data between two networks to evaluate the accuracy of our proposed model. The results show that the proposed model can achieve high performance with 10% mutual data on all datasets.

2.3 Conclusion

In this chapter, we have proposed a novel collaborative learning framework to address the limitations of current ML-based cyberattack detection systems in IoT networks. In particular, by extracting and transferring knowledge from a network with abundant labeled data (source network), the intrusion detection performance of the target network could be significantly improved (even if the target has very few labeled data). More importantly, unlike most of the current works in this area, our proposed framework could enable the source network to transfer the knowledge to the target network even when they have different data structures, e.g., different features. The experimental results then showed that the accuracy of prediction of our proposed framework was significantly improved in comparison with the state-of-the-art UDL model. In addition, the convergence of the proposed collaborative learning model was also analyzed with various cybersecurity datasets. In future



(a) The percentage mutual information of KDD, NSLKDD and UNSW datasets.



(b) The percentage mutual information of IoT datasets.

Figure 2.6: The illustration of AUC with different percentages of mutual information.

work, we can consider using other effective TL techniques to make TL processes more stable and achieve better performance, especially when the amount of mutual information is very limited.

Chapter 3

Collaborative Learning for Cyberattack Detection in Blockchain Networks

This chapter first introduces a novel intrusion detection dataset named BNaT which stands for Blockchain Network Attack Traffic, created from a real blockchain network in our laboratory, and then proposes an effective decentralized collaborative machine learning framework to detect cyberattacks in the blockchain network. Specifically, to develop BNaT, we first set up and implement a blockchain network in our laboratory using Ethereum (an open-source blockchain software) and perform intensive experiments to generate blockchain data (including both normal and malicious traffic data). The main objectives of producing BNaT dataset are fourfold. First, we collect the BNaT in a laboratory environment to have "clean" data samples (i.e., to ensure that the obtained data is not corrupted, erroneous, or irrelevant), that is especially important for training ML models. Second, the BNaT can be easily extended to include new kinds of blockchain attacks, e.g., 51% or double spending attacks. Third, we perform experiments with real attacks in the considered blockchain network, and thus the BNaT can reflect better the actual attack behavior of network than simulations or by artificial attack data generated by GAN in the literature, e.g., [48]. Fourth, we collect the data in different blockchain nodes to have a complete view of effects when the attacks are performed in a decentralized manner. After that, we develop a highly-effective collaborative learning framework to make it more effective in deploying in blockchain networks to detect attacks. In particular, in our proposed learning framework, working nodes in the blockchain network (e.g., mining nodes) can be used as learning nodes to collect blockchain

data (e.g., observing its own traffic and classifying data).

The rest of this chapter is organized as follows. Section 3.1 first describes the fundamentals of blockchain and our proposed model. Section 3.2 then presents our proposed collaborative learning attack detection in detail. After that Section 3.4 presents the simulation and experimental results of this work. Finally, the conclusion and future work are given in Section 3.5.

3.1 Blockchain Network: Fundamentals and Proposed Network Model

3.1.1 Blockchain

Blockchain is a digital ledger technology that provides a transparent, tamper-proof, and secure environment for transmitting data. This technology enables various parties to join, verify, and record transactions without a trusted third party (e.g., a bank). In a blockchain network, multiple nodes are used to simultaneously process and store data. In particular, when a node in the blockchain network receives transactions (e.g., money exchange in the Bitcoin network), it will gather all the transactions and put them in a block. This node will then start a mining process to find a "nonce" value for this block. It is important to note that thanks to the feature of the hash function, there is only a small set of satisfying nonce values for a block, and these values can only be found through an intensive searching process [65]. This mining process is a special process of blockchain networks to provide proofs for validated blocks, and thus this tamper-proof can significantly enhance security for blockchain networks. After the node finds the nonce value for the mining block, this new block will be broadcast and verified by other nodes in the network. Finally, if this block is verified, it will be put into the chain (linked to the hash value of the previous block inside its header). After the block is added to the chain, it is nearly

impossible to change information in this block, and thus this property can guarantee the immutability of the blockchain. Another aspect of blockchain is traceability due to the infeasible collision of the hash function, and thus any transaction or block can be tracked correctly. In summary, blockchain can be termed as a decentralization, immutable, traceable, and time-stamped digital data chain (ledger).

3.1.2 Designed Blockchain Network at our Laboratory

To launch a blockchain network, there are two main kinds of blockchain nodes namely fullnode and bootnode. First, fullnodes take responsibility to store the ledger, participate in the mining process, and verify all blocks and states. Furthermore, they can be used to serve the network and provide data on request, e.g., netstats, which is a visual interface for tracking Ethereum network status (e.g., the block number, mining status, and the number of pending transactions). Second, bootnode is a lightweight application used for the Node Discovery Protocol. The bootnodes do not synchronize blockchain ledger but help other Ethereum nodes discover peers to set up Peer-to-Peer (P2P) connections in the network.

The system model together with essential components of our designed blockchain network is set up as illustrated in Figure 3.1. Specifically, the system includes K fullnodes which are used to receive transactions, mining blocks, and keep the replica of ledger. These nodes continuously synchronize their ledgers together by the P2P protocol with equal permissions and responsibilities for processing data [65]. In order to connect them together, a management node, known as bootnode, is set up. The fullnodes connect and interrogate this bootnode for the location of potential peers in the blockchain network. After being connected, each fullnode can collect data (i.e., transactions) from its network. Transactions can come from different blockchain applications such as cryptocurrency, smart cities, food supply chains, and IoT. First, when transactions are sent to a fullnode, they will be verified and

packed into one block. After the node finds the nonce value for this block, it will broadcast the block together with this nonce value to other nodes in the network for verification. Finally, if the block is verified by the majority of nodes in the network, it will be added to the chain.

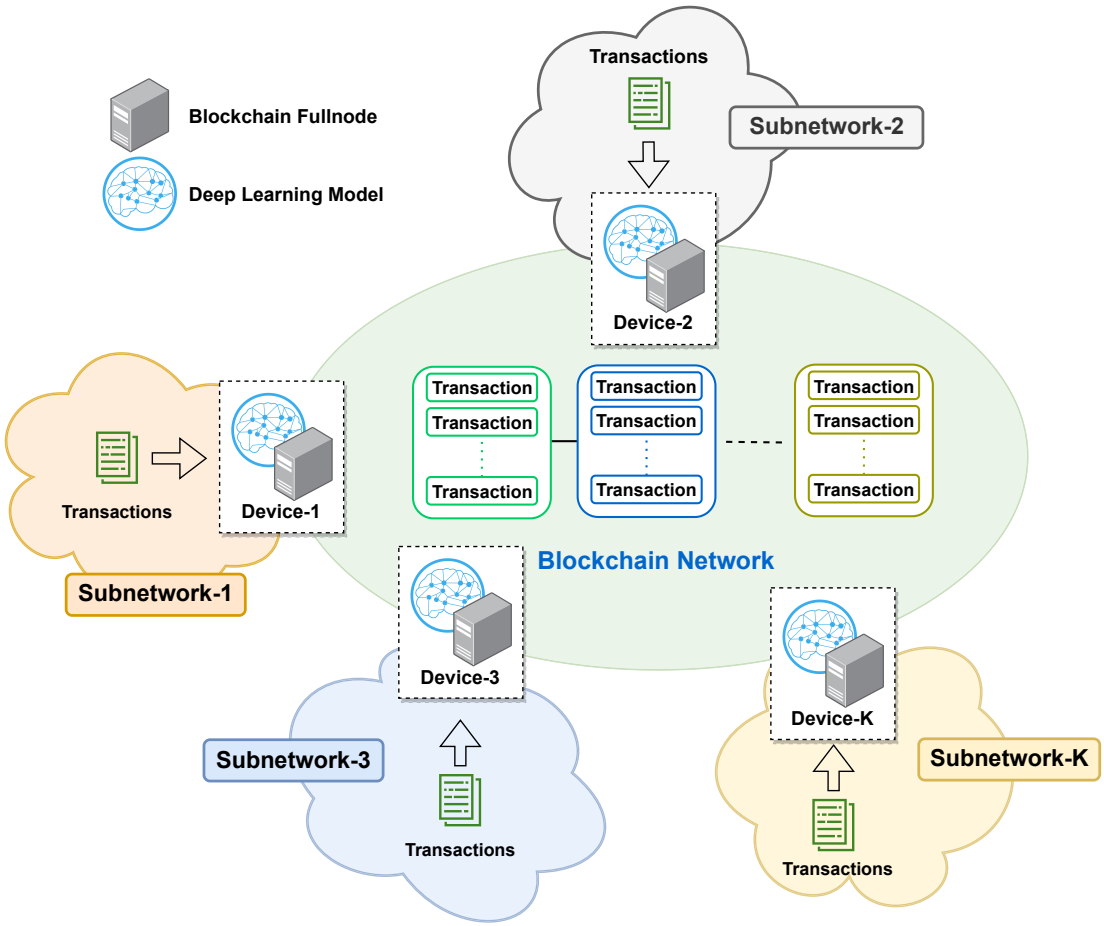


Figure 3.1: Our proposed learning model for blockchain network.

At our laboratory, we design a private blockchain network based on the Ethereum blockchain network. This network also uses the Proof-of-Work (PoW) consensus mechanism, but the block confirmation time is significantly faster than the older version of Bitcoin. More precisely, it takes approximately 15 seconds to validate a new block, representing only 2.5% of the block validation time in Bitcoin [66].

Furthermore, the smart contract layer of Ethereum is suitable for flexible purposes of decentralized environments as mentioned above. In addition, at each node, various attacks, that can cause serious damage to the public blockchain network, will be considered. We then capture the traffic data to analyze their impacts on the blockchain network. In order to capture traffic data of these attacks, we build a dataset collection tool, named BC-ID, which inherits the core of an open-source utility named "kdd99_feature_extractor" [67] and our new designs to fit the considered Ethereum network, i.e., correct the service of packets related to Ethereum nodes, remove meaningless features, and automate label dataset samples based on various given properties. Note that, in practice, there is no software that supports automatically capturing the blockchain network traffic so far. Thus, we analyze the blockchain network traffic data using software named Wireshark [68] and build a new collection tool, namely BC-ID. In this way, we can observe the effects of these attacks on different nodes in the blockchain network.

3.2 Proposed collaborative learning model for intrusion detection in blockchain network

Figure 3.1 describes our proposed framework for intrusion detection in the blockchain network. In our proposed collaborative learning model, the fullnodes in the blockchain networks will be used as Learning Nodes (LNs) to learn knowledge from their collected data inside their subnetworks and share their learned knowledge to improve learning performance for the whole network. We also propose to use a deep neural network at each LN to learn useful information from its collected data. Then, the LNs will share their trained learning models with the CS. After that, the CS will calculate the aggregated model (i.e., the global model) and share this model back with the LNs. When an LN receives this aggregated model from the CS, it will integrate with its current LN and train its local dataset. This process will be repeated

until convergence or reaching a predefined maximum number of iterations. In this way, we can obtain the global learning model for all the LNs.

In our proposed model, each blockchain node has a set of local collected data, and we propose a deep neural network (DNN) using Deep Belief Network (DBN) [19] to learn knowledge from this data. The DBN is a type of deep neural network that is used as a generative model for both labeled and unlabeled data. Therefore, unlike other supervised deep neural networks which use labeled data to train the neural networks (e.g., convolutional neural networks [13]), the DBN has two stages in the training process. The first stage is the pre-training process where the DBN is trained using an unlabeled dataset. The second stage is the fine-tuning process where the DBN is trained using a labeled dataset. Thereby, the DBN can better represent the characteristics of the labeled dataset, and thus it can classify the normal behavior and different types of attacks with high accuracies. In addition, the DBN includes multiple Restricted Boltzmann Machine (RBM) layers for latent representation [19]. In the DBN training process, the current layer generates a latent representation by using the latent representation of the previous layer as the input. Unlike other deep neural networks which can also process both labeled and unlabeled data (e.g., autoencoder deep learning network [13]), the DBN optimizes the energy function of each layer to have better a latent representation of data on each RBM layer in each iteration. Therefore, the DBN is more appropriate to analyze the network traffic where the samples and features have relative coherence with each other.

The whole processes of DBN are illustrated in Figure 3.2. Like other DNNs, the structure of DBN has three components: an input layer, an output layer, and multiple hidden layers. As can be seen in Figure 3.2, the Gaussian Restricted Boltzmann Machine (GRBM) layer – a type of RBM that can process real values of data – is the input layer to receive and transform the input data into binary values. We denote by $k \in \{1, ..., K\}$ the number of learning nodes in the collaborative learning

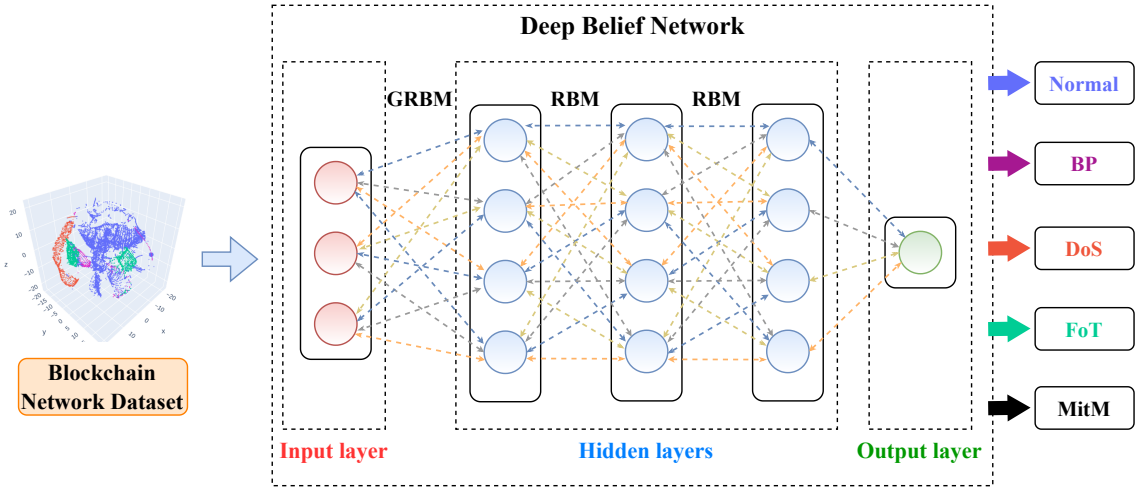


Figure 3.2: The structure of classification-based for intrusion detection learning model in a blockchain network.

model, $\boldsymbol{v}^k$ and $\boldsymbol{h}^k$ to be the vectors of visible and hidden layers of DBN in LN-k, respectively. In addition, M and N are the numbers of visible and hidden neurons of GRBM. We denote by h_n^k, $h_n^k \in \boldsymbol{h}^k$, and v_m^k, $v_m^k \in \boldsymbol{v}^k$, the hidden layer-$n$ and visible layer-m of LN-k. As defined in equation (1.4), the energy function of GRBM of LN-k is calculated as follows:

$$E_G^k(\boldsymbol{v}^k, \boldsymbol{h}^k) = \sum_{m=1}^{M} \frac{(v_m^k - b_{1,m})^2}{2\epsilon_m^2} - \sum_{m=1}^{M}\sum_{n=1}^{N} w_{m,n} h_n^k \frac{v_m^k}{\epsilon_m} - \sum_{n=1}^{N} b_{2,n} h_n^k, \tag{3.1}$$

where $w_{m,n}$ is the weight between visible and hidden neurons; $b_{1,m}$ and $b_{2,n}$ indicate the bias of visible and hidden neurons, respectively; and ϵ_m represents the standard deviation of the neuron in the visible layer. From the result of equation (3.1), we can find the probability that is used in the visible layer of GRBM as in equation (1.5):

$$p_G^k(\boldsymbol{v}^k) = \frac{\sum_{\boldsymbol{h}^k} e^{-E_G(\boldsymbol{v}^k, \boldsymbol{h}^k)}}{\sum_{\boldsymbol{v}^k}\sum_{\boldsymbol{h}^k} e^{-E_G(\boldsymbol{v}^k, \boldsymbol{h}^k)}}. \tag{3.2}$$

Then, we use the probability in equation (3.2) to calculate the gradients of each

GRBM layer with the expectation value $\left\langle . \right\rangle$ as in equation (1.6):

$$\begin{aligned} \nabla g^k_{G,m,n} &= \frac{\partial \log p^k_G(\boldsymbol{v}^k)}{\partial w_{m,n}} \\ &= \left\langle \frac{1}{\epsilon_m} v^k_m h^k_n \right\rangle_{dataset} - \left\langle \frac{1}{\epsilon_m} v^k_m h^k_n \right\rangle_{model}. \end{aligned} \tag{3.3}$$

Next, the gradient of GRBM layers can be calculated:

$$\nabla g^G_k = \sum_{m=1}^{M} \sum_{n=1}^{N} \nabla g^k_{G,m,n}. \tag{3.4}$$

In the next stage, we need to calculate the energy function and the gradient of RBM layers. We denote by M' and N' the numbers of visible and hidden neurons of RBM layers. As defined in equation (1.8) the energy functions of RBM layer of LN-k are defined as follows:

$$E^k_{RBM}(\boldsymbol{v}^k, \boldsymbol{h}^k) = -\sum_{m=1}^{M'} b_{1,m} v^k_m - \sum_{m=1}^{M'} \sum_{n=1}^{N'} w_{m,n} v_m h^k_n - \sum_{n=1}^{N'} b_{2,n} h^k_n. \tag{3.5}$$

Similar to the GRBM layers, we can calculate the gradient of each RBM layer as follows:

$$\nabla g^k_{R,m,n} = \left\langle v^k_m h^k_n \right\rangle_{dataset} - \left\langle v^k_m h^k_n \right\rangle_{model}. \tag{3.6}$$

As a result, the gradient of RBM layers in LN-k can be defined as follows:

$$\nabla g^R_k = \sum_{m=1}^{M'} \sum_{n=1}^{N'} \nabla g^k_{R,m,n}. \tag{3.7}$$

After learning with multiple GRBM and RBM layers, we define $\boldsymbol{X}^{g,r}_k$ as the output of the last hidden layer of LN-k. In this paper, the output layer utilizes the softmax regression function to classify the data samples based on probability. We denote by $\boldsymbol{W}^o$ and $\boldsymbol{b}^o$ the weight matrix and bias vector between the output and

the last hidden layer, respectively. We then can define the probability of the output Z belonging to Class-t as follows:

$$p_k^o(Z = t|\boldsymbol{X}_k^{g,r}, \boldsymbol{W}^o, \boldsymbol{b}^o) = softmax(\boldsymbol{W}^o, \boldsymbol{b}^o), \tag{3.8}$$

where $t \in \{1, .., T\}$ is a class of the output, and T refers to the total classes (including different types of attacks and normal behavior). The prediction $\boldsymbol{Z}_k$ of the probability p_k^o in LN-k can be calculated:

$$\boldsymbol{Z}_k = \underset{t}{\text{argmax}}[p_k^o(Z = t|\boldsymbol{X}_k^{g,r}, \boldsymbol{W}^o, \boldsymbol{b}^o)], \tag{3.9}$$

where Z is the output prediction. Then, we can calculate the gradient between the output layer and the last hidden layer from equation (3.8) as follows:

$$\nabla g_k^o = \frac{\partial p_k^o(Z = t|\boldsymbol{X}_k^{g,r}, \boldsymbol{W}^o, \boldsymbol{b}^o)}{\partial \boldsymbol{W}^o}. \tag{3.10}$$

After that, the results of equation (3.4), equation (3.7), and equation (3.10) are used to calculate the total gradient ∇g_k^t of DBN with multiple GRBM, RBM layers and the output layer of LN-k as follows:

$$\nabla g_k^t = \nabla g_k^G + \nabla g_k^R + \nabla g_k^o. \tag{3.11}$$

In the training process, the DBN first trains its neural network with unlabeled data for pre-training. Then, DBN uses its labeled data to fine-tune its neural network. At this stage, the DBN of LN-k calculates its gradient ∇g_k^t. After that, this gradient is sent to the CS to create an updated global model for all LNs as illustrated in Figure 3.3. For example, at iteration i, the CS receives gradients from all the K

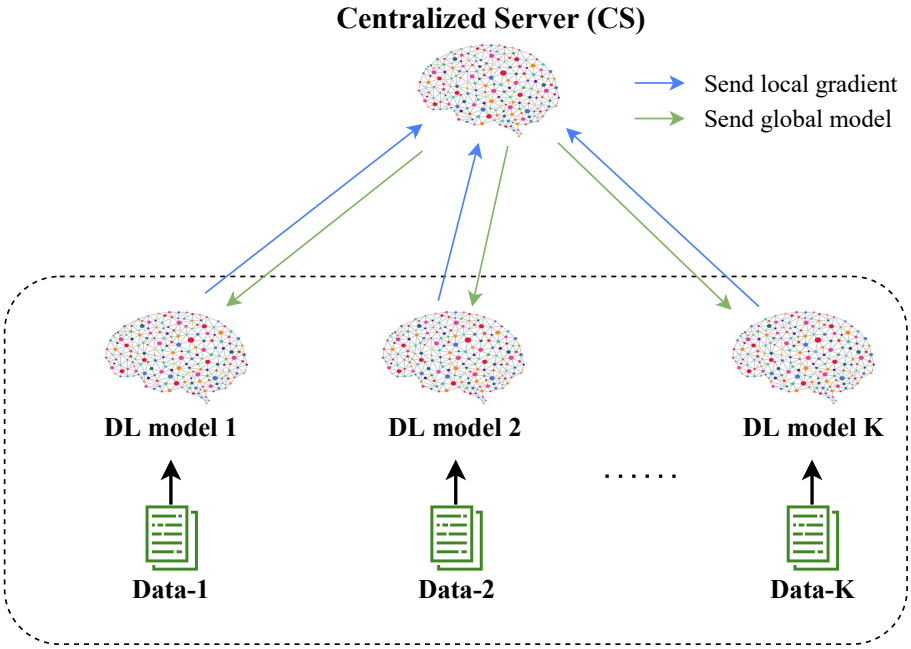


Figure 3.3: The illustration of the collaborative learning between DL models and the CS.

LNs, the CS first performs the average gradient function [69] as follows:

$$\nabla g^* = \frac{1}{K}\sum_{k=1}^{K} \nabla g_k^t. \tag{3.12}$$

We then denote by $\boldsymbol{\varphi}_i$ the global model at iteration i which includes the weight matrix for all layers of the DL model in an LN, and μ represents the learning rate. From the result of equation (3.12), the CS can update the global model at iteration $i+1$ as follows:

$$\boldsymbol{\varphi}_{i+1} = \boldsymbol{\varphi}_i + \mu \nabla g^*. \tag{3.13}$$

Next, the CS sends the latest global model $\boldsymbol{\varphi}_{i+1}$ to the LNs to update their DL models. This process is repeated until it reaches convergence or gets the maximum number of iterations. At this time, we can find the optimal global model $\boldsymbol{\varphi}_{opt}$ that includes the optimal weights of all layers. We denote by $\boldsymbol{W}_{opt}^{o}$ the optimal weight matrix between the output layer and the last hidden layer from $\boldsymbol{\varphi}_{opt}$, the output

Algorithm 3.1 The classification-based collaborative learning algorithm

while i $\leq$ maximum number of iterations or the training process is not converged **do**
  for $\forall k \in K$ **do**
    DBN of LN-k learns $\boldsymbol{X}_k$ and produces $\boldsymbol{Z}_k$.
    Calculate gradient ∇g_k^t.
    Send ∇g_k^t to CS.
  end for
  CS calculates average gradient ∇g^* and global model $\boldsymbol{\varphi}_i$.
  $i = i + 1$.
  CS updates global model $\boldsymbol{\varphi}_{i+1}$.
  CS sends global model $\boldsymbol{\varphi}_{i+1}$ to all LNs.
  LNs update their DBNs.
end while
DBN of LNs predict and classify $\boldsymbol{Z}_k$ from the training dataset $\boldsymbol{X}_k$ with the optimal global model $\boldsymbol{\varphi}_{opt}$.

prediction $\boldsymbol{Z}_k$ of LN-k thus can be calculated as follows:

$$\boldsymbol{Z}_k = \underset{t}{\operatorname{argmax}}[p_k^o(Z = t|\boldsymbol{X}_k^{g,r}, \boldsymbol{W}_{opt}^o, \boldsymbol{b}^o)]. \tag{3.14}$$

Using equation (3.14), the softmax regression of each LN can classify its blockchain network samples to be a normal behavior or a type of attack. Algorithm 4.1 summarizes the process of our proposed collaborative learning model. In our proposed model, the learning model of each network can be trained by the dataset from its local network and exchange learning knowledge with those from other nodes in a blockchain network in an offline manner. In a practical blockchain network with a large number of learning nodes, we can schedule for nodes to exchange the learning knowledge in the offline training phase at an appropriate time to avoid network congestion. In this way, each node can effectively learn knowledge from other nodes while avoiding the traffic congestion of the network. After the training process, the trained models can be used to help nodes to detect attacks in a real-time manner.

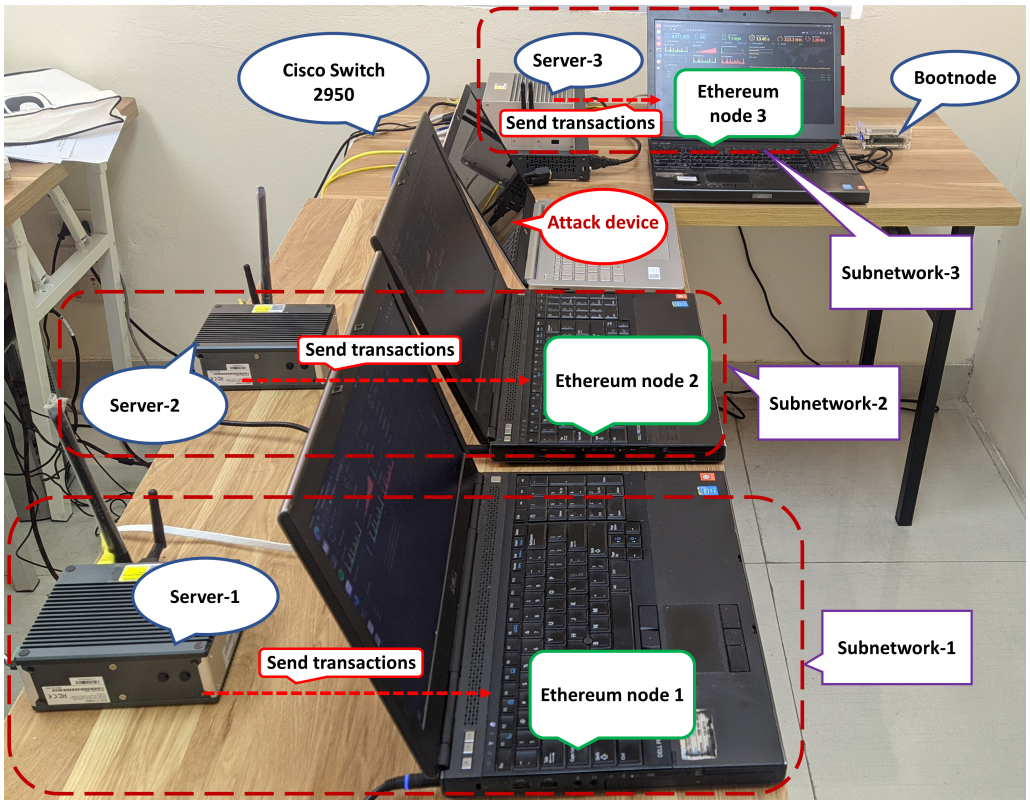


Figure 3.4: Experiment setup.

3.3 Experiment Setup, Dataset Collection and Evaluation Method

This section will explain more details about experiment setup, data collection, and feature extraction over our designed blockchain system.

3.3.1 Experiment Setup

In our experiments, we set up an Ethereum blockchain network in our laboratory which includes three Ethereum fullnodes, an Ethereum bootnode, and a netstats server. All these nodes are connected to a Cisco Switch Catalyst 2950 as illustrated in Figure 3.4. The details of these nodes are as follows:

- Ethereum fullnodes are launched by *Geth v1.10.14* [70] - open-source software for implementation of the Ethereum protocol. These nodes share the same ini-

tial configuration of genesis block, i.e., PoW consensus mechanism, 8,000,000 gas for block gas limit, initial difficulty 100,000. Each node runs on a personal computer with processor Intel® Core™ i7-4800MQ @2.7 GHz, RAM of 16 GB.

- Bootnode is also created by *Geth v1.10.14* and connected to the three Ethereum nodes.

- Ethereum netstats is launched by an open-source software named "eth-netstats" on Github [71].

The normal traffic data is configured with the three trustful servers, while an attack device will execute abnormal/malicious activities to the blockchain network traffic. Each trustful server takes responsibility for generating data and sending transactions to the corresponding Ethereum node in its subnetwork as visualized in Figure 3.4. In summary, in the normal state, the following tasks are scheduled or randomly occur in the network:

- The servers are scheduled to send transactions.

- The users call functions in the deployed smart contracts to explore the ledger. Besides transaction-related functions, the users also send requests to the Ethereum nodes for tracking their balances or the status of miners. Both of these works are randomly made by HTTP requests to the Ethereum node API (Application Programming Interface).

- Ethereum nodes broadcast transactions and mined blocks to synchronize their ledgers. The packets of bootnode are also included in this field.

- WebSockets and JSON-RPC are used when netstats get information from *Geth* clients.

- HTTP requests and replies to view netstats interface and results of cyberattack detection.

3.3.2 Dataset Collection and Feature Extraction

In this section, we consider to detect attacks in network traffic of a permissionless blockchain system [72, 73]. In general, the goals of an adversary are usually the monetary benefit, e.g., chain splitting, and wallet theft, or stability of the network, e.g., delay and information loss. In this work, we focus on the attacks at the network layer. Attacks at the application layer, e.g., 51%, transaction malleability attacks, timejacking, and smart contract attacks, are out of the scope of this work and can be considered in future work. Specifically, we perform four typical types of network attacks that have been reported in blockchain networks, i.e., the BP for wallet theft; DoS; man in the middle for information loss; and FoT for consensus delay. These are the ubiquitous attacks in the network traffic layer that have caused a number of serious consequences for numerous years. More details are as follows:

- *Password Brute-Force (BP) attack*: is derived from traditional cyberattack when attackers perform such attacks to steal blockchain accounts of users. In this way, the attackers can access the wallets of users and steal their digital assets. Previously, the BP attack on KuCoin caused the loss of up to $281 million [7]. To perform this attack, the attacker retries passwords of an Ethereum public key until it finds out the correct login information.

- *Denial of Service (DoS) attack*: is also another common type of attack in blockchain networks as it can be easily performed to attack blockchain nodes. For such kind of attack, the attackers will launch a huge amount of traffic to a target blockchain node in a short period of time. Consequently, the target node will not be able to work as normal, i.e., mining transactions, and even be suspended. In the real-world, Bitfinex [74] was temporarily suspended due to

such kind of attack. Thus, in our setup, a simple DoS attack is simulated, i.e., an SYN flood attack, by repeatedly sending initial connection request (SYN) packets to an Ethereum node.

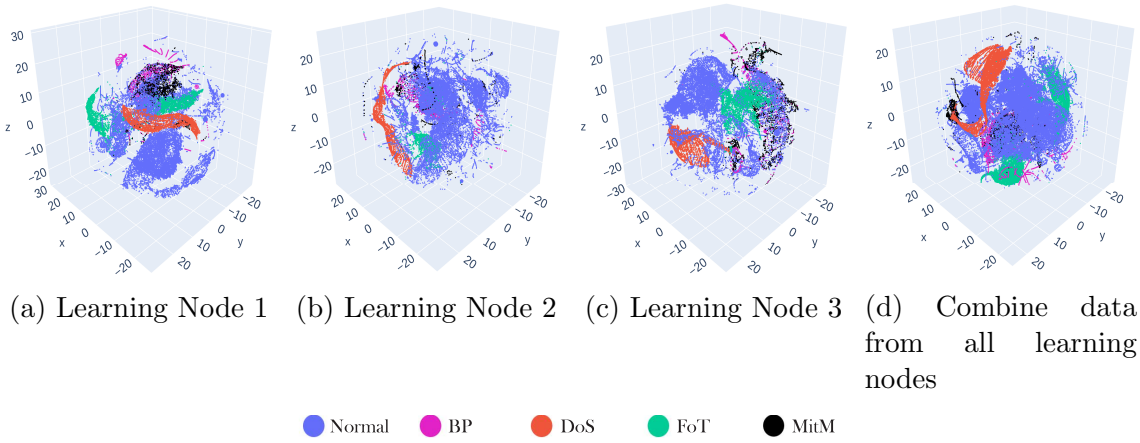
Table 3.1: Features of the designed dataset.

#	Features name	T	Description
Basic features			
1	*duration*	C	length of the connection (seconds)
2	*protocol_type*	D	type of the protocol (i.e., tcp, udp, icmp)
3	*service*	D	network service (e.g., http, ssh, etc)
4	*src_bytes*	C	number of data bytes from source to destination
5	*dst_bytes*	C	number of data bytes from destination to source
6	*flag*	D	normal or error status of the connection
Statistical features			
Features refer to source IP-based Statistical			
7	*count*	C	number of connections to the same source IP as the current connection
8	*srv_count*	C	number of connections to the same service as the current connection
Features refer to these same source IP connections			
9	*serror_rate*	C	% of 'SYN' errors connections
10	*same_srv_rate*	C	% of same service connections
11	*diff_srv_rate*	C	% of different services connections
Features refer to these same service connections			
12	*srv_serror_rate*	C	% of 'SYN' errors connections
13	*srv_diff_host_rate*	C	% of different host connections
Features refer to destination IP-based Statistical			
14	*dst_host_count*	C	number of connections to the same destination IP as the current connection
15	*dst_host_srv_count*	C	number of connections to the same service as the current connection
Features refer to these same destination IP connections			
16	*dst_host_same_srv_rate*	C	% of same service connections
17	*dst_host_diff_srv_rate*	C	% of different services connections
18	*dst_host_same_src_port_rate*	C	% of same both source port and destination IP connections
19	*dst_host_serror_rate*	C	% of 'SYN' errors connections
Features refer to these same service connections			
20	*dst_host_srv_diff_host_rate*	C	% of different host connections
21	*dst_host_srv_serror_rate*	C	% of 'SYN' errors connections

- *Flooding of Transactions (FoT) attack*: targets delay the PoW blockchain network by spamming the blockchain network with null or meaningless transactions. When the number of transactions per second in the Ethereum network suddenly hits the top, a mining node may face two following issues, i.e., too much traffic (similar to that of DoS), and the queue of pending transactions is full. It equates to the unnecessary time burden for the mining process and block propagation [75]. In 2017, the Bitcoin mempool size exceeded 115,000 unconfirmed transactions which led to $700 million worth of transaction stall [73]. In our work, FoT attack is implemented by continuously sending a large number of transactions to an existing smart contract.

- *Man in the Middle (MitM) attack*: is another typical attack where an attacker places himself between the legitimate communicating parties and secretly relays and possibly modifies the information exchanged between them. In this way, the attacker can intercept, read, and modify the blockchain messages. For example, attackers can read the API messages between users and blockchain nodes to steal their wallet password [76]. To implement MitM, an attack device first filters '*eth_sendrawtransaction*' packets, which represent any transaction from users to blockchain nodes. Then, the contents of these packets are randomly modified, leading to invalid transactions.

In order to capture traffic data of these attacks, we build a dataset collection tool, named BC-ID, which inherits the core of an open-source utility named "kdd99_feature_extractor" [67] and our new designs to fit the considered Ethereum network, i.e., correct the service of packets related to Ethereum nodes, remove meaningless features, and automate label dataset samples based on various given properties. To do this, we first use the 'libpcap-dev' library of Linux to capture all the network data (including normal and different types of attacks) from the local net-



(a) Learning Node 1 (b) Learning Node 2 (c) Learning Node 3 (d) Combine data from all learning nodes

Figure 3.5: Visualization using t-SNE for collected datasets.

work. Then, the BC-ID is used to extract features from the collected data, filter the attack samples, and label them as normal or a type of attack. In particular, the BC-ID starts by capturing raw traffic data based on 'libpcap-dev' package of Linux OS. Since each blockchain network has a few specific *ports* for peer connections, client connections, and so on, BC-ID targets to filter and analyze traffic data in these ports. For example, the Ethereum blockchain network uses port 30303 for the TCP port listener, and port 8545 for JSON-RPC by default. Similar to the KDD99 dataset [77], BC-ID extracts features and then separates them into two categories, e.g, basic features (i.e., all the attributes can be extracted from a TCP/IP connection) and traffic features (i.e., statistics of packets with the same destination host or service in a window interval). Especially, our goal is to achieve a trained model that can be applied to our proposed real-time blockchain attack detection system, when the number of samples in a prediction frame is limited. Thus, the BC-ID collects the dataset frames in which each frame lasts for 2 seconds and extracts their features. The BC-ID then puts all collected data in this frame into a single file. Finally, we merge all single files together to make the full dataset. In summary, Table 3.1 shows 21 features in the designed dataset, which are separated into two types, i.e., discrete

Table 3.2: The number of samples in the designed dataset.

Ethereum node / Class	Node-1 (samples)	Node-2 (samples)	Node-3 (samples)
Normal	50,000	50,000	50,000
BP	5,000	5,000	5,000
DoS	5,000	5,000	5,000
FoT	5,000	5,000	5,000
MitM	5,000	5,000	5,000

(D) and continuous (C). For continuous features, they are calculated in 2 seconds time window (similar to that of the famous KDD99 dataset [77]).

In each Ethereum node, the separated dataset is collected in five states (classes), i.e., normal state (Class-0), BP attack (Class-1), DoS attack (Class-2), FoT attack (Class-3), and MitM attack (Class-4). The normal state is captured in two hours, the rest of them in an hour through the designed BC-ID tool. As described above, when a node is attacked, the normal traffic still exists. Therefore, the attack samples can be filtered out by features-based two properties, i.e., the source and destination IP address of the attack device; *service*, *src_length*, and *dst_length* of the samples, which are analyzed by Wireshark [68]. To improve the diversity of the designed dataset, the normal traffic data in the attack state is mixed with traffic data in the normal state. In our experiments, a number of random samples in each state are selected to reduce the size of the bulk dataset as shown in Table 3.2. In fact, we mix normal traffic data in an equal ratio, i.e., 10,000 samples per normal state, normal traffic data at BP, DoS, FoT, and MitM, respectively.

Figure 3.5 illustrates the visualization of our designed dataset using the t-Distributed Stochastic Neighbor Embedding (t-SNE) [78] with three most important components. Although sharing the same configurations for t-SNE, the dataset of each LN has a different distribution in the output. In the 3D view, the DoS and FoT attack samples show a fairly clear separation from normal state points. Otherwise, the

BP and MitM attack samples collide with the normal state samples. This indicates that discriminating BP and MitM samples from the normal data points would be significantly challenging.

3.3.3 Evaluation Method

The confusion matrix with accuracy, precision, and recall proposed in [63] is widely used to evaluate the performance of machine learning algorithms. Let TP, FP, TN, and FN denote "True Positive", "False Positive", "True Negative", and "False Negative", respectively. The accuracy of the total system with T classes including normal behaviors and different types of attacks is as follows:

$$Accuracy = \frac{1}{T}\sum_{t=1}^{T}\frac{\mathrm{TP}_t + \mathrm{TN}_t}{\mathrm{TP}_t + \mathrm{TN}_t + \mathrm{FP}_t + \mathrm{FN}_t}. \tag{3.15}$$

The precision of class t is calculated as $P_{re}^t = \frac{\mathrm{TP}_t}{\mathrm{TP}_t + \mathrm{FP}_t}$. In this chapter, we use weighted average precision to evaluate the performance of the whole system. We denote by S_t the number of samples of class t and S as the number of samples of the whole dataset. The weighted average precision is calculated as follows:

$$Precision = \sum_{t=1}^{T}\frac{P_{re}^t S_t}{S}. \tag{3.16}$$

The recall of class t is calculated by $R_e^t = \frac{\mathrm{TP}_t}{\mathrm{TP}_t + \mathrm{FN}_t}$. The weighted average recall that we use to calculate the performance of the total system is calculated as follows:

$$Recall = \sum_{t=1}^{T}\frac{R_e^t S_t}{S}. \tag{3.17}$$

In the next section, we also use accuracy, precision, and recall to evaluate and compare the performance of our proposed Collaborative Learning model (proposed

Table 3.3: Simulation results.

Model	2 Learning Nodes (LNs)				3 Learning Nodes (LNs)				
	Proposed CoL	CeL	IL		Proposed CoL	CeL	IL		
			LN-1	LN-2			LN-1	LN-2	LN-3
Accuracy	**97.427**	97.330	97.036	96.865	**97.276**	97.270	96.827	96.731	96.825
Precision	**93.861**	93.620	92.793	92.000	**93.448**	93.249	92.209	91.343	92.798
Recall	**93.567**	93.324	92.590	92.162	**93.189**	93.176	92.067	91.829	92.063

CoL) with two other baseline methods, i.e., Centralized Learning model (CeL) and Independent Learning model (IL).

3.4 Experimental Results and Performance Evaluation

In this section, we use the collected datasets of three nodes described in the aforementioned section for the corresponding LNs. The dataset of each LN is randomly split into training and testing datasets. All LNs use DBN with the same structure of neural network for learning and detecting attacks. However, the LNs have to work in different learning models and various scenarios. Each LN has itself training and testing dataset, and thus we can use these datasets to evaluate and compare the performance of the proposed CoL, the CeL, and the IL in different scenarios.

3.4.1 Simulation Results

In this section, we present the simulation results with the dataset of LNs in different learning models. The details of datasets using for simulation are as follows:

- **Proposed Collaborative Learning Model (proposed CoL)**: Each LN learns its training dataset and performs collaborative learning with other LNs to generate the global model. Then, we use the global model to test the merged testing dataset of all participated LNs.

- **Centralized Learning Model (CeL)**: The centralized node (e.g., one of the mining node in the network) is assumed to be able to collect data from all the

nodes in the network and train the DL model on the collected datasets. Then, we use the trained model to test data based on the merged testing dataset of all participated LNs.

- **Independent Learning Model (IL)**: Each LN learns its training dataset without sharing knowledge with other LNs. Then, we use this model to test data based on the merged testing dataset of all participated LNs.

3.4.1.1 Convergence Analysis

Figure 3.6 describes the convergence of the proposed CoL, the CeL, and the IL (in terms of accuracy) of three LNs in the training process. The proposed CoL is obtained at the LN-1 after obtaining the global model. The CeL has a large number of training samples from three LNs so it can reach the convergence with around 97% accuracy after 400 epochs. Besides, the proposed CoL and IL converge after 800 epochs and 1300 epochs, respectively. After 3,000 learning epochs, we can observe that the proposed CoL has a higher accuracy compared with that of the IL (i.e., 97.2% vs 96.8%). The reason is that the proposed CoL can obtain the exchange knowledge from DL models of other LNs. Thereby, it can achieve a similar performance as that of the CeL.

3.4.1.2 Performance Analysis

Table 3.3 presents the simulation results in two cases, i.e., two participated LNs, and three participated LNs. In both cases, we can observe the same trend when the accuracy, precision and recall of the proposed CoL are higher than those of the IL and nearly equal to those of the CeL. In particular, the accuracy of the proposed CoL is higher than that obtained by LN-1 in IL (approximately 0.5%), and the precision of the proposed CoL is about 2% higher than that obtained by LN-2 in IL in the case of three participated LNs. These results demonstrate that the proposed

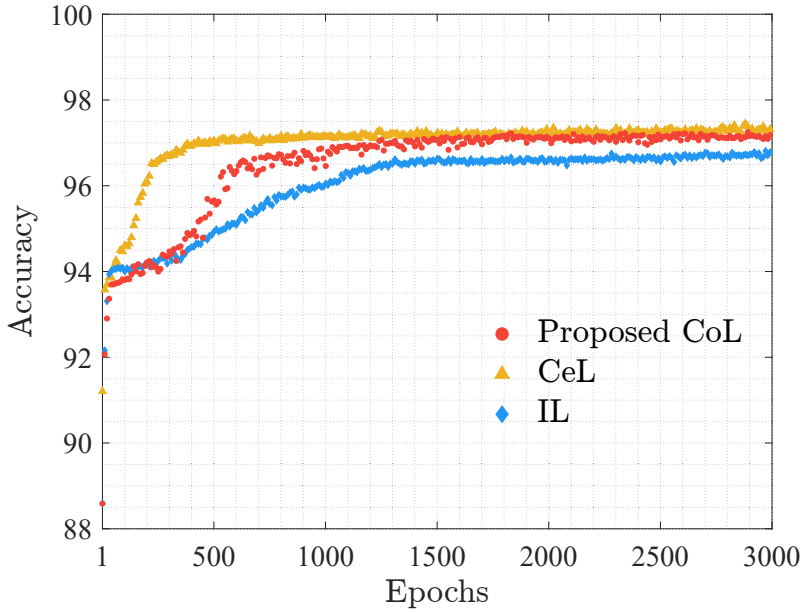


Figure 3.6: Training process of considered learning models.

Table 3.4: Real-time experimental results of 3 LNs models.

Model	2 Learning Nodes (LNs)				3 Learning Nodes (LNs)					
	Proposed CoL		CeL		Proposed CoL			CeL		
	LN-1	LN-2	LN-1	LN-2	LN-1	LN-2	LN-3	LN-1	LN-2	LN-3
Accuracy	**98.611**	**98.242**	98.464	98.097	**98.440**	**98.131**	**97.686**	98.503	98.192	97.771
Precision	**97.433**	**96.871**	97.146	96.634	**97.146**	**96.717**	**95.902**	97.138	96.679	95.864
Recall	**96.529**	**95.606**	96.159	95.243	**96.101**	**95.328**	**94.214**	96.258	95.481	94.427

CoL can exchange knowledge with other LNs to improve its ability of detection, so it can achieve better performance in classifying attacks in the blockchain network than those of the IL. It also demonstrates that the learning model of IL should not be used to classify the data of other LNs. In addition, without sharing the LN datasets with a central node for training (e.g., a cloud server), the proposed CoL can achieve nearly the same accuracy as those of the CeL in all the scenarios.

3.4.2 Experimental Results

In this section, we present the experimental results obtained through real-time experiments at our laboratory. In this experiment, each blockchain node is established as a learning model to become an LN. Each LN learns its local dataset and

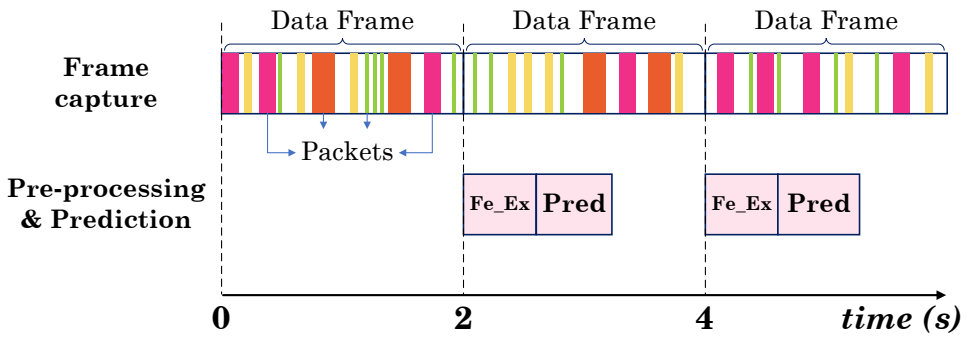


Figure 3.7: Timeline of verification phase.

then performs real-time attack detection in the blockchain network. We consider the scenario of two LNs and three LNs with the proposed CoL and the CeL. In the training process, the proposed CoL and the CeL are fed with similar datasets as explained in the previous section. We then implement the trained model to all the participated LNs to perform real-time attack detection for both learning models in the testing process.

3.4.2.1 Real-time capturing and processing

In a real-time system, the cyberattack detection system continuously receives a number of the Ethereum network traffic data. Therefore, the system has to perform capturing, collecting frames, extracting features, analyzing and predicting within a period of time, i.e, 2 seconds. Figure 3.7 shows the timeline of the cyberattack detection program. The data frame is exploited by our feature extractor function (Fe_Ex) of BC-ID tool, and this is input for the trained model to predict (Pred) and classify packets to be normal or attack. All processes have to complete in 2 seconds before the next data frame of IP packets coming. To verify the predicted results from the trained model, all frames and prediction results are stored. These frames are merged into a full validation dataset and labeled by own designed BC-ID. After that, these ground truth labels are compared with the prediction results to obtain a validation report.

3.4.2.2 Performance Analysis

Table 3.4 presents the experimental results of the proposed CoL and the CeL with different participated LNs. We obtain the same trends as those of the simulation results. The accuracy results obtained by two and three learning models of both proposed CoL and CeL are slightly higher than those of the simulations at about 1%. This is because each type of attack has attack sample distributions within a period of time. Table 3.5 presents the number of samples of each class collected in 15 minutes. In this table, we can observe that Class-1 and Class-4 have small numbers of samples during this period, this can lead to a low accuracy in statistics for these classes and reduce the total accuracy of the model. However, our proposed CoL still has better performance than those of the CeL in LN-1 in the case of two LNs (i.e., up to 0.2% accuracy, 0.3% precision, and 0.4% recall). Overall, our proposed CoL always achieves the best performance with approximately 98.6% accuracy, 95.43% precision, and 96.52% recall with two LNs and 98.44% accuracy, 97.14% precision and 96.1% recall with three LNs. These results demonstrate that our proposed CoL can detect attacks with nearly the same accuracies for all participated LNs as those of CeL.

3.4.2.3 Real-time Monitoring and Detection

Figure 3.8 illustrates the real-time monitoring of our proposed CoL for normal state and three types of attacks in the network. Figure 3.8(a) is the normal state (Class-0) of the network with a high number of normal samples and a low number of attack samples. Then, the BP and MitM attacks are performed. Figure 3.8(b) and Figure 3.8(e) show a slight increase in the number of BP attacks and MitM attacks. This is because the number of BP attack samples is much smaller than other states in a period of time as in Table 3.5. In this case, the detection mechanism is activated and alarms the network under the BP attack (Class-1). Similarly, the

Table 3.5: The number of samples on LN-1 in five hours.

	Class-0	**Class-1**	**Class-2**	**Class-3**	**Class-4**
Number of samples	736,897	2,424	481,532	886,389	3,483
Portion (%)	34.912	0.115	22.814	41.994	0.165

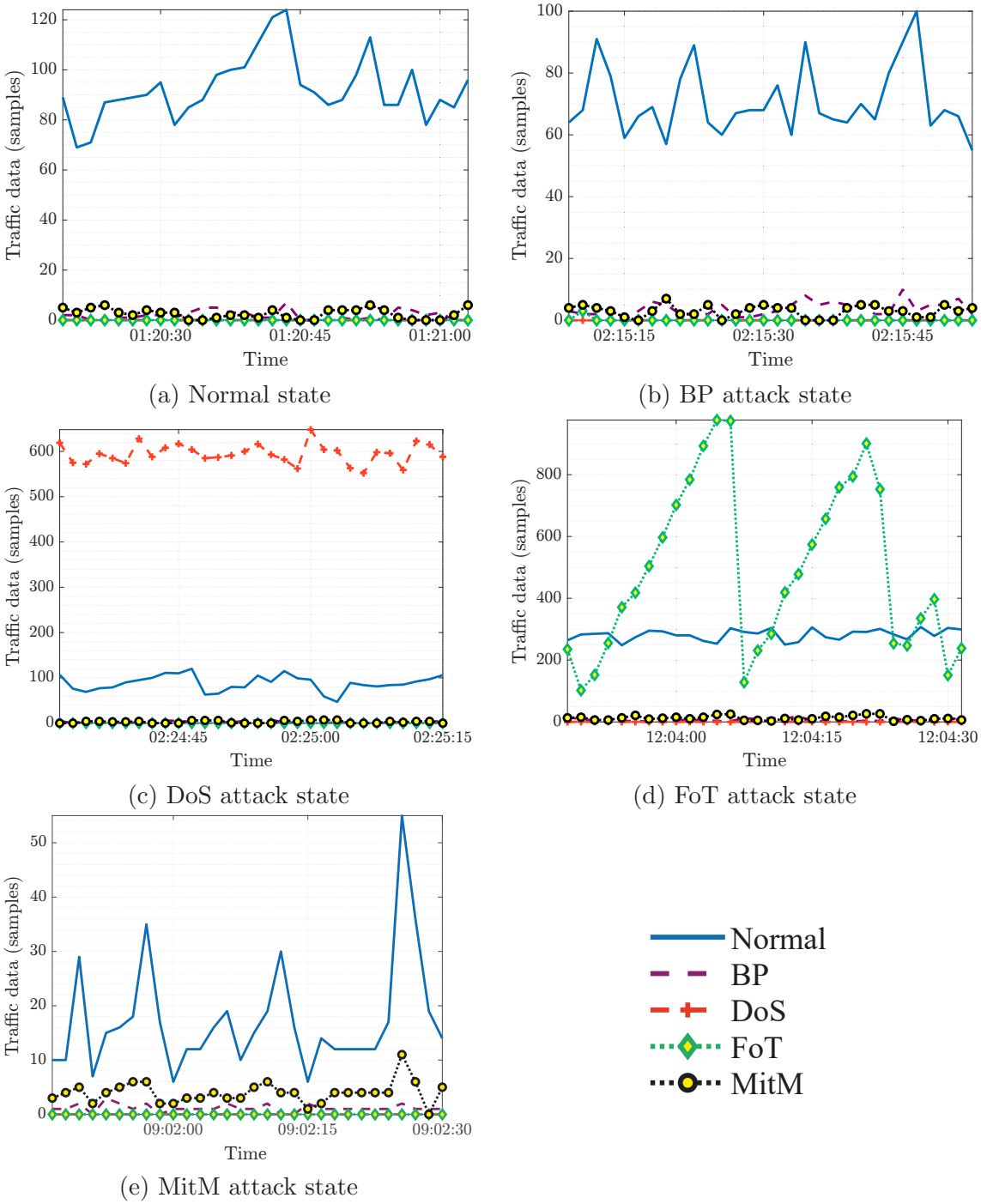
DDoS attack in the network is described in Figure 3.8(c) with a high increase in the number of samples of DoS attacks. Finally, Figure 3.8(d) describes the FoT attacks. Unlike other attacks, the FoT attacks include a large number of samples, thus it increases both the number of normal and attack samples (above 200 traffic samples per 2 seconds) more than other attacks (about 100 traffic samples per 2 seconds). Thereby, in all the cases, we can observe that our proposed intrusion detection system can detect attacks effectively in a real-time manner.

3.4.2.4 Real-time Processing Capacity

In this experiment, we fix the number of input samples in our proposed model to find the maximum real-time processing capacity in capturing and detecting attacks. Figure 3.9 illustrates the real-time processing capacity of our proposed model. The processing time τ is counted from the time when our proposed model reads the file containing the samples, until completing classification and producing the output. This work is repeated 20,000 times to determine the stability of the detection time of our proposed model. We vary the number of input samples multiple times to find the appropriate number that is adapted to the condition in Figure 3.7. In most of the cases (98%), our proposed framework can classify 85,000 samples in less than 2 seconds. These results demonstrate that our proposed detection framework is efficient to deploy to detect attacks in real-world blockchain networks. It can not only detect attacks with high accuracy (up to 98.6%) but also quickly (up to 85,000 samples within 2 seconds).

3.5 Conclusion

In this chapter, we have proposed a novel collaborative learning framework for a cyberattack detection system in a blockchain network. First, we have implemented a private blockchain network in our laboratory. This blockchain network was used to (1) generate data (both normal and attack data) to serve the proposed learning models and (2) validate the performance of our proposed learning framework in real-time experiments. After that, we have proposed a highly-effective learning model that allows to be effectively deployed in the blockchain network. This learning model enables nodes in the blockchain to be actively involved in the detection process by collecting data, learning knowledge from their data, and then exchanging knowledge together to improve the attack detection ability. In this way, we can not only avoid problems of conventional centralized learning (e.g., congestion and single point of failure) but also protect the blockchain network right at the edge. Both simulation and real-time experimental results then have clearly shown the efficiency of our proposed framework. In the future, we plan to continue developing this dataset with other emerging types of attacks and develop more effective methods to protect blockchain networks.



(a) Normal state

(b) BP attack state

(c) DoS attack state

(d) FoT attack state

(e) MitM attack state

Figure 3.8: Real-time blockchain cyberattack detection: The proposed CoL model of 3 LNs in Ethereum node 1.

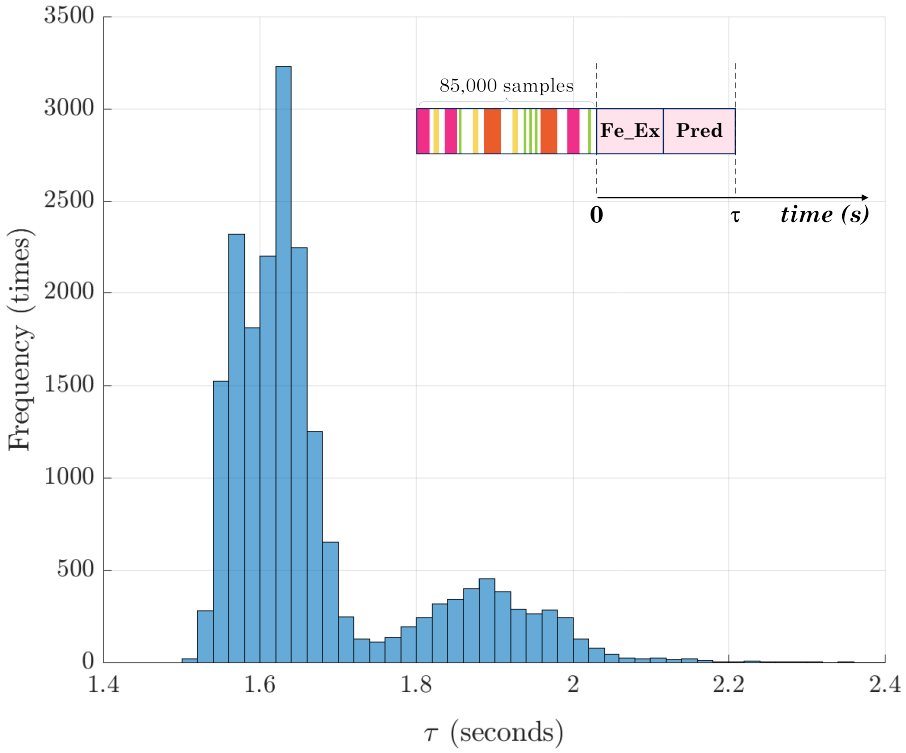


Figure 3.9: Histogram of real-time detection duration.

Chapter 4

Collaborative Learning for Detection of Attacks to Transactions and Smart Contracts

In this chapter, we first set up experiments in our laboratory to deploy various kinds of attacks on transactions and SCs in a blockchain system (i.e., a private Ethereum system). First, we collect all the transactions in Ethereum Nodes (ENs) to build a dataset, called **Attacks on Blockchain Transactions Dataset (ABTD)**. To the best of our knowledge, at the time we perform this work, this is the first cyberattack dataset on transactions and SCs in a blockchain network synthesized in a laboratory. To enrich the dataset, we create a large number of individual accounts to send transactions to the blockchain network for execution randomly. This dataset can be used for both research and industry purposes to address cyberattacks in transactions and smart contracts. In addition, to deal with the challenge in analyzing Bytecode from transactions, we propose a novel ML-based framework that analyzes transactions and SCs without the need of understanding the SC source codes. Our proposed framework automatically extracts transaction features in real-time and efficiently analyzes them to detect insight attacks. To do this, we first build a highly-effective tool, called **Blockchain Code Extraction and Conversion Tool (BCEC)**, to convert important information of transactions and SCs to grey images. This tool calls a transaction using a transaction hash (i.e., a feature of the transaction) and then extracts key fields like Bytecode and value from the transaction. After that, it can convert the contents into images for further processing. Second, we develop an ML-based approach based on CNN to learn and detect attacks insight transactions and SCs. To the best of our knowledge, **this is the first ML-based frame-**

work that analyzes the Bytecode directly and detects various types of attacks in transactions and SCs. Such an ML-based framework, which uses important information from transactions for analysis, is more flexible and easier to detect new types of attacks than other vector-based methods. To address the third challenge about centralized attack detection, we develop a novel collaborative cyberattack detection framework that can detect cyberattacks inside transactions and SCs in real-time with high accuracy. In our proposed framework, the CNN of each Ethereum node can exchange learning knowledge (i.e., the trained models) with other nodes to create a global model. In this way, the learning model of each node can improve the detection accuracy without sending their local data over the network.

The rest of this chapter is organized as follows. Section 4.1 describes the overview of our proposed model. Section 4.2 then discusses in detail our proposed collaborative learning attack detection for transactions and smart contracts. After that Section 4.3 provides the simulation and real-time experimental results of this work. Finally, the conclusion and future work are given in Section 4.4.

4.1 Designed Blockchain System and Our Proposed Collaborative Learning Framework

In our laboratory, we set up experiments to collect datasets for training and testing our framework. We first deploy a blockchain system based on a private Ethereum network in our laboratory (more details are shown later in Figure 4.3). This network uses the latest version of the Ethereum network (i.e., Ethereum 2.0). This version uses Proof-of-Stake (PoS) as a consensus mechanism for validating new blocks. PoS is a consensus protocol that verifies the blocks of transactions using the machine of coin owners. In this protocol, the owners pledge their coins as collateral through staking, in exchange for the opportunity to validate blocks and receive

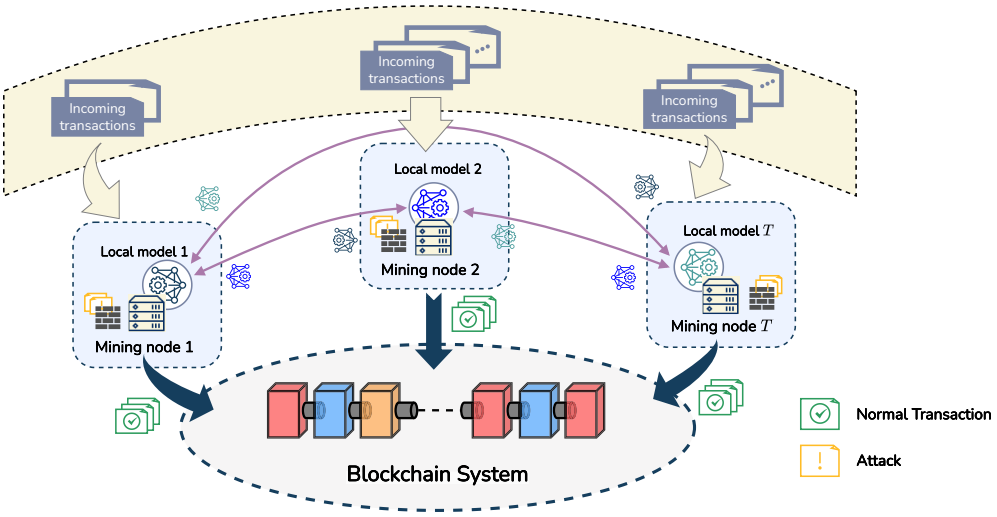


Figure 4.1: The system model of the proposed collaborative learning framework for detection of attacks to transactions and smart contracts. While receiving transactions, our framework will perform preprocessing to extract important information. After that, our collaborative learning will perform the attack detection process to detect network normal behaviour or a type of attack.

rewards. Consequently, PoS requires fewer computational resources compared to Proof of Work. Our system includes various ENs, to collect data from their local networks, and bootnodes, the management nodes to connect ENs together. The ENs can receive transactions from various types of blockchain applications such as smart cities, smart agriculture, IoT, and cryptocurrency. As described above, the transactions are first sent to ENs. They are then put into a block, and the ENs will perform the mining process to put them into the main chain. We perform various attacks using malicious transactions and SCs on this system. These attacks (i.e., DoS with block gas limit, overflows and underflows, flooding of transactions, reentrancy, delegatecall, and function default visibility) happened and caused serious damage to blockchain systems [79]. Through experiments, we build a state-of-the-art dataset with both normal and attacked transactions and SCs to evaluate the performance of attack detection methods.

In this chapter, we consider a blockchain system with T ENs working in a

blockchain system as described in Figure 4.1. When an EN receives transactions from the blockchain network, it uses **BCEC** (the tool that we developed in our laboratory) to preprocess them. This tool is deployed at each EN to extract information from important features, such as Bytecode and Value. It then preprocesses the information and converts this information into grey images for further processing. This tool thus executes these processes in real-time to provide the input for the next real-time processing deep learning model. After that, we propose a collaborative learning framework for analyzing the images to detect insight attacks in transactions and SCs. In our framework, each EN uses its local dataset to train a deep neural network. After the training process, each EN shares its trained model with other nodes and also receives their trained models in return. Afterward, every EN aggregates all the received trained models from other nodes together with its current trained model to generate a new global model for further training (we will explain more details in the next section). In this way, EN can exchange its learning knowledge with the neural network of other ENs. This approach can not only improve the overall learning knowledge of the neural network of all ENs but also protect the privacy of local data over network transmission. By preventing the transmission of the local data of each EN over the network, our approach can also reduce network traffic to avoid network congestion. Thus, the neural networks of ENs can improve the accuracy of detecting attacks for transactions and SCs in blockchain systems.

4.2 Proposed Attack Detection Framework

In our proposed attack detection framework, the ENs are used to learn and share their learning knowledge with others to improve the accuracy of their attack detection. At each EN, we propose to use a deep neural network as a detector to learn the data of the local EN. After that, the EN exchanges its learning knowledge

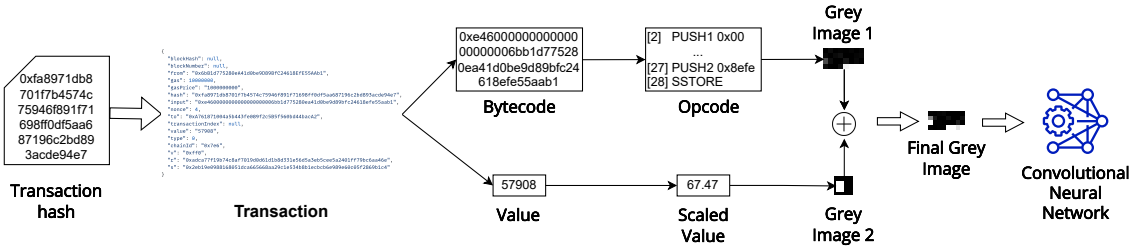


Figure 4.2: The preprocessing process of the proposed collaborative learning framework for detection of attacks to transactions and smart contracts. Our developed BCEC tool first collects the transactions in Ethereum nodes. It then extracts the content of transactions to find "Bytecode" and "Value". After that, this tool converts them into images for further processing.

(i.e., trained model) with other ENs. When an EN receives trained models from others, it will integrate them with its current model to train its local dataset. This process is iteratively repeated until reaching a predefined number of iterations. In summary, our proposed framework includes three processes. The first process is preprocessing. In this process, our proposed framework captures and extracts the important information of the incoming transactions and then converts them to grey images. The second process is to develop a deep convolution neural network to classify the grey images to detect attacks. The last process is collaborative learning. In this process, each EN can exchange the trained model with others to improve the accuracy of attack detection.

4.2.1 Preprocessing Process

Figure 4.2 describes our proposed preprocessing process for transactions in a blockchain system. The main purposes of the preprocessing process are extracting the important features from incoming transactions and converting them into images for further processing. It is worth noting that SCs are a set of agreements to deploy transactions. For implementation, a server has to send transactions of the SCs to the EN for a mining process. From the EN point of view, we only can observe transaction hashes (i.e., the unique addresses of incoming transactions) which are

represented in a series of hexadecimal numbers. The preprocessing process has three steps to deal with these transaction hashes as follows:

- **Step 1:** Capture transaction hashes from the EN and then recover transactions from transaction hashes to have the full information of all features in transactions such as content, value, block hash, block number, chainID, etc.
- **Step 2:** Extract the content of two crucial features in transactions named Bytecode and value. The bytecode feature includes the main functions of transactions and the value feature indicates the amount of ETH (Ethereum) involved in a transaction. Although we can effectively use the bytecode feature in detecting various types of attacks in transactions and SCs, it does not provide any information on various specific types of attacks, such as Flooding of Transactions [80], where the transaction content is null. Thus, it may be inefficient if we only rely on the bytecode feature for analysis. Therefore, we propose to enhance the attack detection framework by incorporating information from the value feature. After that, we apply appropriate preprocessing methods to the corresponding features as follows:
  - **Bytecode feature**: Extract the content and then transform it into opcode using EVM Bytecode Decompiler [81]. The opcode is a series of executed comments in assembly. Thus, we propose to convert all features of this assembly code to a grey image named Grey Image 1.
  - **Value feature**: we first scale its content to an appropriate range and then convert it to another grey image named Grey Image 2.
- **Step 3:** In this step, we combine both Grey Image 1 and Grey Image 2 to create the Final Grey Image. This Final Grey Image includes all essential information of a transaction and an SC in the blockchain system. They can

be used to train the deep convolution neural network to find out the hidden attacks inside.

In this framework, all these steps are encapsulated in the **BCEC** tool. This tool can perform the preprocessing process in real-time to support the analysis of collaborative attack detection to detect hidden attacks for transactions and SCs in a blockchain system.

4.2.2 Learning Process

In our proposed framework, at each EN, we implement a detector that can help to detect attacks based on the transformed images from the preprocessing process with high accuracy. The core component of the detector is developed based on a Deep Convolutional Neural Network (CNN). The reason for using CNN is that this framework can classify a large amount of labeled data, especially in image classification with high accuracy [20]. Additionally, the CNN model does not have to learn their local data separately, it can exchange its trained model with other ENs to improve the learning knowledge as well as enhance the accuracy of attack detection. In detail, the architecture of CNN in an EN includes three types of layers, i.e., convolution layer, max pooling layer, and fully connected layer [20]. Figure 1.4 describes the layer of a CNN in an EN. These layers can be described as follows:

- **Convolution layer:** The neurons in this layer are formed in feature maps to learn the feature representation of the input. In addition, these feature maps can connect with others of the previous layer by weight parameters called filter banks [21]. In this layer, the input data is convoluted with weight parameters in every iteration to create feature maps.

- **Max pooling layer:** The main purpose of this layer is to reduce the resolution of feature maps in the previous layer. To do this, this layer selects the largest

values in areas of feature map [20] and then sends them to the next layer.

- **Fully connected layer:** This layer performs classification functions for the neural network. In this layer, the feature maps from previous layers are first flattened. They are then put into a fully connected layer for classification. The softmax function is included at the end of this layer to produce the output prediction.

We denote by $\boldsymbol{D}$ a local dataset of an EN to train a CNN. $\boldsymbol{D}$ includes $\boldsymbol{S}$ images and $\boldsymbol{Y}$ labels so we can denote $\boldsymbol{D} = (\boldsymbol{S}, \boldsymbol{Y})$. We consider $n = \{1, .., N\}$ as the training layer of the neural network. We denote by N the number of training layers of the neural network. We denote by $\boldsymbol{I}$ the matrix features of image $\boldsymbol{S}$, and $\boldsymbol{I_i}$ as the matrix features of image $\boldsymbol{S}$ at iteration i. The output of a convolution layer n, $n \in \{1, .., N\}$, at iteration $i + 1$ can be calculated as in equation (1.15):

$$\boldsymbol{I}_{n+1,i} = \gamma_n\Big(\boldsymbol{I}_{n,i} * \boldsymbol{F}_n\Big), \tag{4.1}$$

where $(*)$ is the convolutional operation, γ_n is the activation function and $\boldsymbol{F}_n$ is the filter bank of layer n. After that, the output of the convolution layer is put into a max pooling layer. The output of a max pooling layer can be calculated as follows:

$$\boldsymbol{I}_{n+2,i} = \varphi\Big(\boldsymbol{I}_{n+1,i}\Big), \tag{4.2}$$

where φ is the max pooling function that selects the maximum value in a pooling area. We denote by $\boldsymbol{I}_{e,i}$ the matrix features of the last image after processing with multiple convolution layers and max pooling layers. $\boldsymbol{I}_{e,i}$ is put into a softmax function to classify and produce the output in the fully connected layer. We consider $l \in \{1, ..., L\}$ as the classification group number, $\hat{Y}_l \in \hat{\boldsymbol{Y}}$ as the output prediction, the probability that an output prediction $\hat{Y}$ belongs to group l can be calculated as

follows:

$$\begin{aligned} p(\hat{Y}_l = l|\boldsymbol{I}_{e,i}, \boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i}) &= softmax(\boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i}) \\ &= \frac{e^{\boldsymbol{W}_{e,i}\boldsymbol{I}_{e,i}+\boldsymbol{b}_{e,i}}}{\sum_l e^{\boldsymbol{W}_{e,l,i}\boldsymbol{I}_{e,i}+\boldsymbol{b}_{e,l,i}}}, \end{aligned} \tag{4.3}$$

where $\boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i}$ are the weights and biases of the fully connected layer at iteration i, respectively; and $\boldsymbol{W}_{e,l,i}, \boldsymbol{b}_{e,l,i}$ as weights and biases of the fully connected layer at iteration i to classify an output prediction into class l. Based on equation (4.3), we can calculate a vector of prediction $\hat{\boldsymbol{Y}}$ which includes output prediction $\hat{Y}_l$ belonging group l with probability p as follows:

$$\hat{\boldsymbol{Y}} = \underset{l}{\text{argmax}}[p(\hat{Y}_l = l|\boldsymbol{I}_{e,i}, \boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i})]. \tag{4.4}$$

In this stage, we compare the output predictions with the labels using a sparse categorical cross-entropy function to calculate the loss for backpropagation. We denote by $Y_l \in \boldsymbol{Y}$ the label of class l in $\boldsymbol{Y}$. The loss function can be calculated as follows:

$$\boldsymbol{J}(\boldsymbol{W}) = -\sum_{l=1}^{L} Y_l \log \hat{Y}_l. \tag{4.5}$$

We denote by $\boldsymbol{W}$ the model of the neural network. Based on equation (4.5), we can calculate the gradient of this function as follows:

$$\nabla\boldsymbol{\theta} = \frac{\partial \boldsymbol{J}(\boldsymbol{W})}{\partial \boldsymbol{W}} = -\frac{\partial\Big(\sum_{l=1}^{L} Y_l \log \hat{Y}_l\Big)}{\partial \boldsymbol{W}}. \tag{4.6}$$

After having the gradient based on equation (4.6). We then use it for the Adam optimizer to update the parameters of the neural networks. We consider m_{i+1} and v_{i+1} as the moment vectors of the next iteration $i+1$ of the Adam optimizer. The m_{i+1} and v_{i+1} can be calculated from the gradient and Adam functions [82] as

$m_{i+1} = A_1(\nabla\boldsymbol{\theta})$ and $v_{i+1} = A_2(\nabla\boldsymbol{\theta})$. We denote by $\boldsymbol{\Gamma}_i$ a trained model, and θ_i a global model at iteration i. With β_{i+1} as the learning rate, a new trained model at the next iteration $i+1$ can be calculated as follows:

$$\begin{aligned}\boldsymbol{\Gamma}_{i+1} &= \boldsymbol{\Gamma}_i - \beta_{i+1}\frac{m_{i+1}}{\sqrt{v_{i+1}}} \\ &= \boldsymbol{\Gamma}_i - \beta_{i+1}\frac{A_1(\nabla\boldsymbol{\theta}_i)}{\sqrt{A_2(\nabla\boldsymbol{\theta}_i)}}.\end{aligned} \tag{4.7}$$

4.2.3 Collaborative Learning Process

In this chapter, we propose a Collaborative Deep Convolutional Neural Network framework (Co-CNN) to detect the different types of attacks in a blockchain network. In this framework, each EN has a CNN model to train and test its dataset. The CNN model can receive trained models from other ENs to improve the accuracy of attack detection. To do this, the CNN model of an EN first gets the trained model (gradient) based on equation (4.6). It then sends the trained model to other ENs and receives trained models from others. We denote by T the total number of ENs and $t \in T$ as the EN number. We consider at iteration i, an EN receives $T-1$ trained models from others. $\boldsymbol{\theta}_{t,i}$ is the trained model of EN t at iteration i. It can aggregate all trained models using the following formula [83]:

$$\boldsymbol{\theta}_{i+1} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\theta}_{t,i}, \tag{4.8}$$

where $\boldsymbol{\theta}_{i+1}$ is the new aggregated trained model. After generating a new aggregated trained model, each EN will calculate a new trained model using equation (4.7). This process continuously repeats until the algorithm converges or reaches the predefined maximum number of iterations. After the training process, we can obtain the optimal trained model in each EN to analyze and detect the attacks inside a series of grey images. This process is summarized in Algorithm 4.1.

Algorithm 4.1 The learning process of Co-CNN model

1: **while** $i \leq$ maximum number of iterations **do**
2: **for** $\forall t \in T$ **do**
3: The CNN of the EN-t learns D_t to produce $\hat{Y}$.
4: The EN-t creates the trained model θ_t and sends it to others.
5: The EN-t receives $T-1$ trained models from others.
6: EN calculates a new optimal trained model $\mathbf{\Gamma}_{i+1}$.
7: **end for**
8: $i = i + 1$.
9: **end while**
10: EN uses its optimal trained model $\mathbf{\Gamma}_{optimal}$ to detect attacks based on input grey images.

4.3 Performance Analysis

4.3.1 Experiment Setup

In our experiments, we set up an Ethereum 2.0 system in our laboratory as shown in Figure 4.3. This version of Ethereum uses a new consensus mechanism namely Proof-of-Stake (PoS) instead of Proof-of-Work (PoW). There are five Ethereum nodes, two bootnodes, a trustful device, and an attack device in our experiments. All these devices are connected to a Cisco switch, which serves as the central hub for our local network. The configuration of these devices is as follows:

- Ethereum nodes are launched by *Geth v1.10.22* - an official open-source implementation of Ethereum network [70] and *Prysm v3.2.0* - an official implementation of the PoS consensus mechanism in Ethereum 2.0 [84]. They share the same genesis configurations, e.g., chainID, block gas limit at 30,000,000 gas, etc. The configurations of nodes 1, 2, and 3 are workstation computers with processor Intel Core i9-10900 @5.2 GHz, RAM of 64 GB. The configurations of nodes 4 and 5 are personal computers with processor Intel Core i7-4810MQ @3.8 GHz, RAM of 16 GB.

- *Geth* bootnode and *Prysm* bootnode are also created by *Geth v1.10.22* and

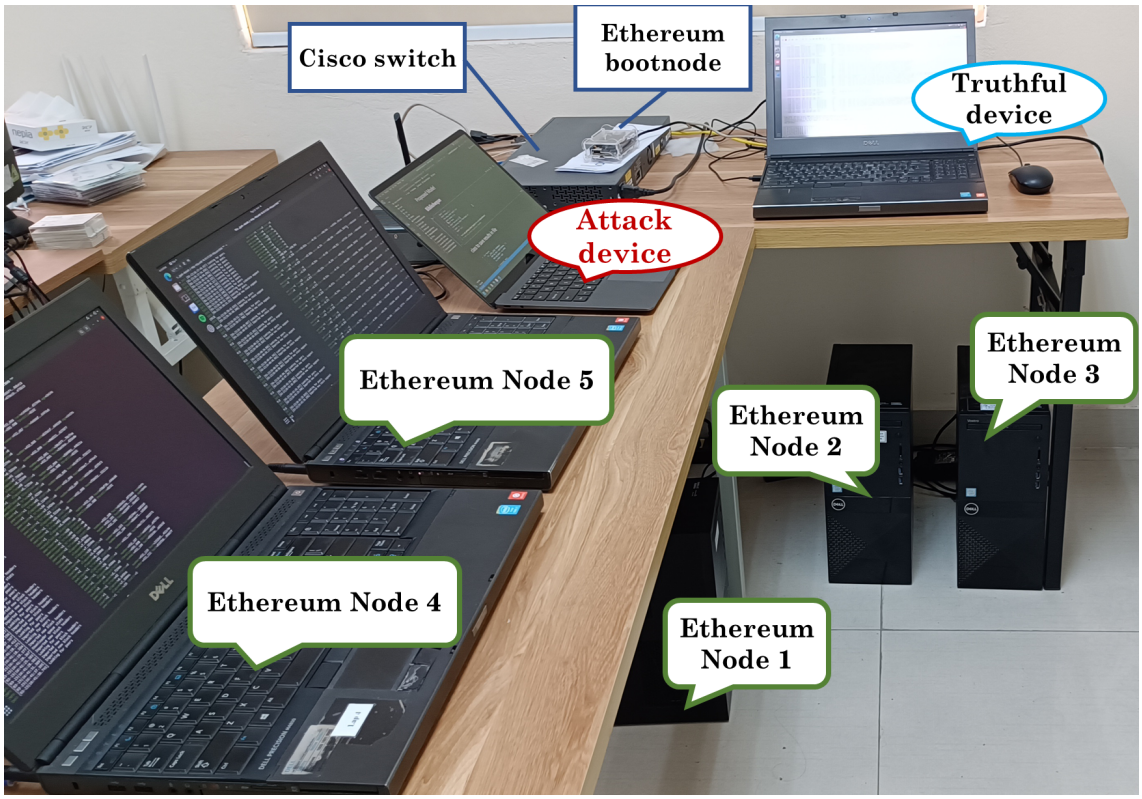


Figure 4.3: Real experiment setup.

Prysm v3.2.0, respectively. They are responsible for connecting all the Ethereum nodes together.

4.3.2 Dataset Collection

According to the detailed analysis of the public Ethereum network on transaction behavior [85], the addresses that are associated with less than 10 transactions account for 88% of total addresses. About 50% received addresses appear only one time for a transaction in history. This is because most people want to create transactions anonymously. Therefore, to create diversity and reality for our dataset, we need to create a large number of unique accounts (i.e., 10,000 accounts in our experiments) to send transactions to Ethereum nodes. A truthful server, as shown in Figure 4.3, randomly selects accounts from these accounts to create transactions for the blockchain system.

4.3.2.1 Normal State

For the normal state, we use *OpenZeppelin Contracts* [86] library as the secured SCs. Two types of transactions below are used to generate samples randomly for the normal state.

- Exchange ETH: On the public Ethereum network, most transactions only exchange the ETH to another address without any bytecode. This kind of transaction accounts for 75% of the total samples of the normal state in our experiment.
- Transactions-related SCs: There are two types of these transactions. The transactions for deploying SCs and the transactions that interact with functions in deployed SCs. We perform three essential categories of SCs in the Ethereum system, i.e., Tokens/Coins/NFT, Ethereum 2.0 deposit, and SCs for other purposes.

Although the number of original SCs is minuscule compared to the total transactions in the dataset. The content of transactions and deployed SCs are not duplicated. Because we randomly select not only the senders and recipients but also the amount of ETH and inputs of functions in any generated transaction.

4.3.2.2 Attack States

SCs have a number of vulnerabilities listed in SWC [87] because of programmers, consensus mechanisms, and compilers. Attackers can exploit these weaknesses of SC to perform attacks and then steal money in blockchain systems [79]. In this work, we regenerate several real-world attacks from the tracks that they left on the ledger of Ethereum. We give a brief description of the six types of application layer-based attacks.

- *DoS with Block Gas Limit (DoS)*: There are several functions inside SCs. These functions can be temporarily disabled when their gas requirements exceed the block gas limit. A *DoS* case occurred in 2015 when the 1,100 ETH jackpot payout of SC GovernMental was stuck [87]. The GovernMental SC is deployed in our work, and we continuously join the jackpot to disable the payout function.

- *Overflows and Underflows (OaU)*: In solidity language, if a variable is out of its range, it is in the overflow or underflow state. In this case, the variable is turned to another value (e.g., 0 for overflow and $2^{256}-1$ for underflow). Attackers can use this vulnerability to bypass SCs' conditions when withdrawing funds. For example, they can bypass the requirements of checking their accounts' balances. Several real *OaU* attacks were detected, e.g., 2^{256} BEC tokens, CSTR token, $800k USD of PoWH token [88], and so on [87]. We re-perform the above *OaU* attacks on their original SCs in the dataset.

- *Flooding of Transactions (FoT)*: Attackers spam a number of meaningless transactions to delay the consensus of blockchain networks. Such an attack caused the unconfirmation of 115k Bitcoin transactions in 2017 [80]. In our setup, *FoT* attacks are generated by continuously sending a negligible amount of ETH from a random sender to another arbitrary recipient.

- *Re-entrancy (Re)*: When the SCs do not update their states before sending funds, attackers can recursively call the withdraw function to drain the SCs' balances. Two types of *Re* attacks are single-function and cross-function. The single-function type happened and led to a loss of 3.6 million ETH in 2016. Both types of Re are performed in our dataset [87].

- *Delegatecall (DeC)*: *delegatecall()* is the mechanism to inherit functions, storage, and variables from other deployed SCs. If the inherited SCs are attacked,

Table 4.1: Number of samples on the proposed ABTD dataset.

Class	**Number of samples**	**Portion (%)**
Normal	152,423	50.34
DoS	22,994	7.59
OaU	29,254	9.66
FoT	41,732	13.78
Re	22,682	7.49
DeC	22,455	7.41
FDV	11,209	3.73
Total	302,749	100

they will in-directly affect the main SC. To implement, we re-create the 2^{nd} Parity MultiSig Wallet attack [87]. In this attack, attackers took control and suicide the inherited SC.

- *Function Default Visibility (FDV)*: If the programmers do not define the visibility of functions in SCs, it will default to the public. Thus, anyone can interact with those functions. For implementation, we perform the 1^{st} Parity MultiSig Wallet attack [87]. In this attack, attackers took control of this SC through an FDV flaw.

Table 4.1 shows the number of samples in each class of our proposed dataset. The proportions of the samples in the classes are not balanced, e.g., the number of *Re* samples is twice that of *FDV*. Because *Re* requires a series of attack transactions instead of only one attack transaction as in *FDV*.

4.3.3 Evaluation Methods

The confusion matrix [63, 64] is widely used to evaluate the performance of machine learning models. We denote by TP, TN, FP, and TN "True Positive", "True Negative", "False Positive", and "True Negative". In this chapter, we use ubiquitous parameters (i.e., accuracy, precision, recall) in the confusion matrix to evaluate

the performance of models. The accuracy of a model can be calculated as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{4.9}$$

In addition, we use the macro-average precision and macro-average recall to evaluate the performance of the models. With L as the number of classification groups (i.e., the total number of normal and attack states), the macro-average precision is calculated as follows:

$$\text{Precision} = \sum_{l=1}^{L} \frac{\text{TP}_l}{\text{TP}_l + \text{FP}_l}. \tag{4.10}$$

The macro-average recall of the total system can be calculated as follows:

$$\text{Recall} = \sum_{l=1}^{L} \frac{\text{TP}_l}{\text{TP}_l + \text{FN}_l}. \tag{4.11}$$

4.3.4 Simulation and Experimental Results

In this section, we present the simulation and real-time experimental results of our experiments. In particular, we use the confusion matrix to evaluate our proposed model's performance (in terms of accuracy, precision, and recall) compared to the centralized model.

In this section, we compare our proposed model in two schemes. We use our proposed preprocessing process in the first scheme as in Figure 4.2. In the second scheme, we eliminate the value feature and use only the Bytecode preprocessing to analyze the transactions and SCs. Though the results of these schemes, we demonstrate the efficiency of our proposed preprocessing process in combining various features of transactions. We use CNN to classify different types of cyberattacks and normal behavior in transactions and SCs. Figure 4.4 describes the accuracy results of two schemes. In this figure, the model w/-V has accuracy, precision, and recall

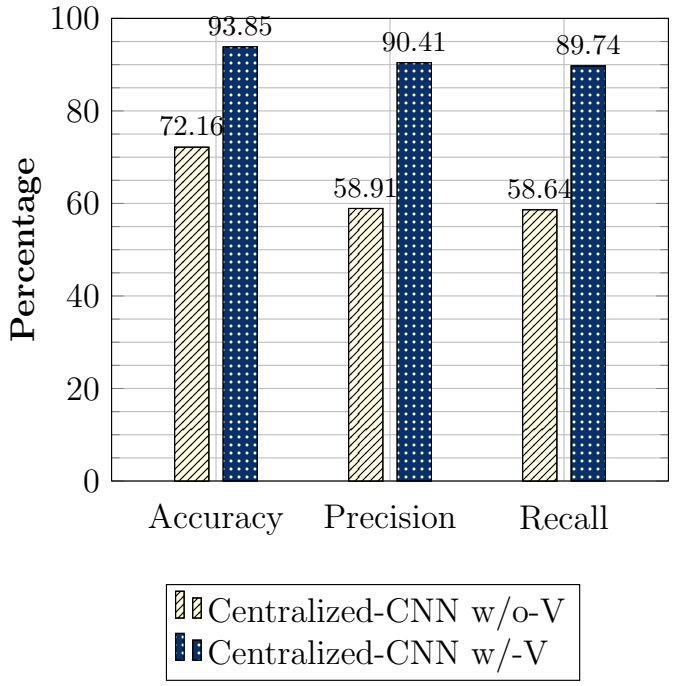


Figure 4.4: The results of the preprocessing processes in different schemes.

at 93.849%, 90.413%, and 89.742%, respectively. These results outperformed the model w/o-V which has accuracy, precision, and recall at 72.163%, 58.911%, and 58.638%, respectively. Especially, Figure 4.5 provides detailed information for all types of attacks and normal behavior. In Figure 4.5, we can see that the model w/o-V cannot detect DoS and FoT attacks because it classifies all samples of DoS and FoT attacks into normal behavior. In contrast, the model w/-V can detect these types of attacks with high accuracy at about 97% for DoS detection and 100% for FoT detection. This is because the value feature is essential to support the learning models to detect various types of important attacks.

4.3.4.1 Accuracy Analysis

In this section, we perform experiments to compare the performance results of the centralized model with our proposed model. The centralized model (Centralized-CNN) that we design can learn knowledge from all ENs for training and testing pro-

cesses. Besides, we use different schemes of the collaborative learning model with 3 Ethereum nodes (Co-CNN-3), 5 Ethereum nodes (Co-CNN-5), and 10 Ethereum nodes (Co-CNN-10). In each scheme, the collected datasets are divided equally among all Ethereum nodes. To implement experiments, we first perform cyberattacks on transactions and SCs in our deployed private Ethereum platform to collect datasets from all ENs. In our proposed collaborative learning model, each EN uses its local dataset for both training and testing processes. However, in the training process, the ENs can exchange their trained models with others to improve their learning knowledge as well as the accuracy of attack detection. On the other hand, in the Centralized-CNN, all the local datasets of ENs will be gathered into a big dataset for its training and testing process.

The performance results of two scenarios of preprocessing processes (i.e., without value feature (w/o-V) and with value feature (w/-V) with all schemes are also provided in Table 4.2 and Table 4.3. Table 4.2 presents the performance of the simulation results of all schemes with the w/o-V preprocessing process. In Table 4.2, the accuracy, precision, and recall are nearly the same at around 72-73%, 58-59%, and 58-59%, respectively. In contrast, in Table 4.3, we can observe that the performance of all schemes with the w/-V preprocessing process outperforms those w/o-V preprocessing process at about 93-94%, 90-91%, and 89-90% in accuracy, precision, and recall, respectively. In detail, we first can see in Table 4.3 that the performance results of our proposed models are nearly the same as the Centralized-CNN. However, in various ENs such as EN-5 of the Co-CNN-5, the accuracy, precision, and recall are higher than those of the Centralized-CNN at around 0.6%, 0.6%, and 0.7%, respectively. Specifically, Figure 4.6 provides detailed information for each type of attack of the Centralized-CNN and EN-5 of Co-CNN-5. These figures show that the misdetection of EN-5 of the Co-CNN-5 is dramatically reduced compared to the Centralized-CNN. In detail, the misdetection of the EN-5 from Normal to DoS is at

Table 4.2: Simulation results w/o-V with Centralized-CNN, Co-CNN-3, Co-CNN-5, and Co-CNN-10 models.

	Centralized-CNN	Co-CNN-3			Co-CNN-5				
		EN-1	EN-2	EN-3	EN-1	EN-2	EN-3	EN-4	EN-5
Accuracy	72.163	71.686	71.761	72.080	72.735	72.519	72.211	72.760	72.627
Precision	58.911	58.323	58.298	58.646	59.676	59.300	58.818	59.699	59.032
Recall	58.638	57.539	57.951	58.608	58.955	58.807	58.415	59.444	58.969

	Co-CNN-10									
	EN-1	EN-2	EN-3	EN-4	EN-5	EN-6	EN-7	EN-8	EN-9	EN-10
Accuracy	72.768	73.333	73.184	73.117	73.150	72.984	73.017	73.267	73.516	73.117
Precision	58.169	59.462	59.107	58.957	58.779	58.621	58.288	59.503	59.013	59.125
Recall	58.131	58.531	58.462	58.727	58.775	58.285	58.528	59.066	59.192	58.650

Table 4.3: Simulation results w/-V with Centralized-CNN, Co-CNN-3, Co-CNN-5, and Co-CNN-10 models.

	Centralized-CNN	Co-CNN-3			Co-CNN-5				
		EN-1	EN-2	EN-3	EN-1	EN-2	EN-3	EN-4	EN-5
Accuracy	93.849	93.88	94.384	94.115	94.347	94.057	94.148	94.206	94.439
Precision	90.413	90.216	91.162	90.860	90.794	90.540	90.637	90.903	91.029
Recall	89.742	89.665	90.688	89.970	90.329	89.932	90.025	90.514	90.536

	Co-CNN-10									
	EN-1	EN-2	EN-3	EN-4	EN-5	EN-6	EN-7	EN-8	EN-9	EN-10
Accuracy	93.633	94.248	93.849	93.566	93.899	93.832	93.516	93.732	93.699	93.849
Precision	89.326	90.611	90.095	89.969	90.106	90.048	89.252	90.684	89.778	90.464
Recall	89.206	89.716	89.313	89.114	89.745	89.289	89.213	89.464	89.298	89.477

0.88% which is smaller than that of the Centralized-CNN at 1.14%. Similarly, the misdetection of the EN-5 from OaU to Normal is at 0.926% of total samples of OAU which is smaller than that of the Centralized-CNN at 3.89%.

4.3.4.2 Convergence Analysis

In this section, we compare the convergence of different models, i.e., the Centralized-CNN, and the collaborative model with 3, 5, and 10 Ethereum nodes. Figure 4.7 describes the accuracy and loss of these models in 1,000 iterations. In general, all of the models converged after about 800 iterations in terms of accuracy and loss. While the accuracies of Centralized-CNN, Co-CNN-3, and Co-CNN-5 models fast reach the convergence after 400 iterations at about 93%, the accuracies of Co-CNN-10 need about 800 iterations to converge and reach 93%. The same trends happen

with the loss. This is because the number of samples of each EN in Co-CNN-10 is much smaller than those of other models while the number of workers is higher than those of other models. Thus, Co-CNN-10 needs more time to exchange learning knowledge with other models. It finally reaches convergence after about 800 iterations and has accuracies nearly the same as other models.

4.3.4.3 Real-time Attack Detection

In this section, we consider a practical scenario by evaluating the performance of the system in real-time cyberattack scenarios. To do this, we first take the trained models from all schemes (noted that the trained modes are trained in the schemes as in the accuracy analysis, i.e., Centralized-CNN, Co-CNN-3, Co-CNN-5). There are 5 blockchain nodes participating in these experiments and they join a private Ethereum network as described in the above section. After the learning models are trained, they are deployed on ENs. In the experiments, both two cases with value and without value preprocessing processes are considered. In real-time scenarios, both normal and attack samples continuously come to the blockchain node. Thus, the BCEC has to collect all the transaction traffic in 3 seconds into a package and then convert them into images. All processes including preprocessing (i.e., converting samples into images) and processing (i.e., model prediction) must be completed within 3 seconds before the next package comes.

Table 4.4 presents the performance of Co-CNN-3, Co-CNN-5, and Centralized-CNN models in two cases of preprocessing. In general, we can observe in Table 4.4(a) that the performance of these models in accuracy, precision, and recall w/-V in the preprocessing process is at about 88-91%, 76-80%, and 77-79%, respectively. These results outperform those of the w/o-V in preprocessing process with accuracy, precision, and recall at about 65-66%, 44-51%, and 48-51%, respectively. In addition, when we compare the same case w/-V in preprocessing process of the simulation as in

Table 4.4: Real-time experiment results.

(a) Centralized-CNN and Co-CNN w/-V

	Centralized-CNN					Co-CNN-3					Co-CNN-5				
	EN-1	**EN-2**	**EN-3**	**EN-4**	**EN-5**	**EN-1**	**EN-2**	**EN-3**	**EN-4**	**EN-5**	**EN-1**	**EN-2**	**EN-3**	**EN-4**	**EN-5**
Accuracy	89.603	89.542	89.668	89.702	89.291	88.663	88.582	88.655	88.794	88.471	90.928	90.896	90.957	91.061	90.614
Precision	76.851	75.806	76.956	76.690	75.582	76.755	75.872	76.845	77.191	75.912	80.192	78.835	80.469	80.846	78.576
Recall	76.858	77.117	76.888	76.767	76.822	78.523	78.939	78.531	78.563	78.724	78.870	79.044	78.757	78.762	78.747

(b) Centralized-CNN and Co-CNN w/o-V

	Centralized-CNN					Co-CNN-3					Co-CNN-5				
	EN-1	**EN-2**	**EN-3**	**EN-4**	**EN-5**	**EN-1**	**EN-2**	**EN-3**	**EN-4**	**EN-5**	**EN-1**	**EN-2**	**EN-3**	**EN-4**	**EN-5**
Accuracy	65.877	65.780	65.643	66.312	65.734	66.804	66.640	66.569	67.115	66.797	65.606	65.579	65.512	66.212	65.830
Precision	47.263	46.105	46.739	47.544	46.012	51.442	50.024	51.116	51.641	50.433	44.668	44.078	44.534	44.994	44.141
Recall	51.383	51.576	51.434	51.318	51.427	49.670	49.888	49.586	49.557	49.625	48.447	48.544	48.381	48.372	48.518

Table 4.3 and the real-time experimental results as in Table 4.4(a), we can observe that the accuracy, precision, recall of the real-time experimental results are little smaller than those of simulation results about 3%, 10%, and 11%, respectively. This is because, in simulation, we implement multiple types of attacks on the blockchain system and then collect data to have enough samples for the dataset to train the model. However, in real-time scenarios, various attack types, such as Re, DeC, and FDV, rarely appear during the experiment. Thus, it makes more difficult for the learning models to detect them in real-time.

Specifically, we can observe in Table 4.4(a) that EN-4 of Co-CNN-5 has higher performance in accuracy, precision, and recall than EN-4 of the Centralized-CNN about 1.3%, 4%, and 2%, respectively. Therefore, in real-time detection scenarios, our proposed model still demonstrates better performance in detecting attacks than in simulation.

4.3.4.4 Real-time Monitoring and Detection

Figure 4.8 shows the real-time cyberattack monitoring from the output of our proposed model Co-CNN-5 in Ethereum node 1. In these figures, the normal and each type of attack are displayed in different lines. Figure 4.8(a) displays the normal state of the system with the high value of the predicted normal state over time. We

can observe that in the normal state, the predicted states of all types of attacks are nearly 0. When a type of attack happens, the predicted state of that attack will increase, e.g., the FoT attack state as in Figure 4.8(d). As described in the previous section, in real-time scenarios, RE, Dec, and FDV attack states have a little number of attack samples. Therefore, their predicted states in Figure 4.8(b), Figure 4.8(f) and Figure 4.8(g) do not have high values. However, our proposed model can still detect all of the attacks in real-time with high accuracy at 91%.

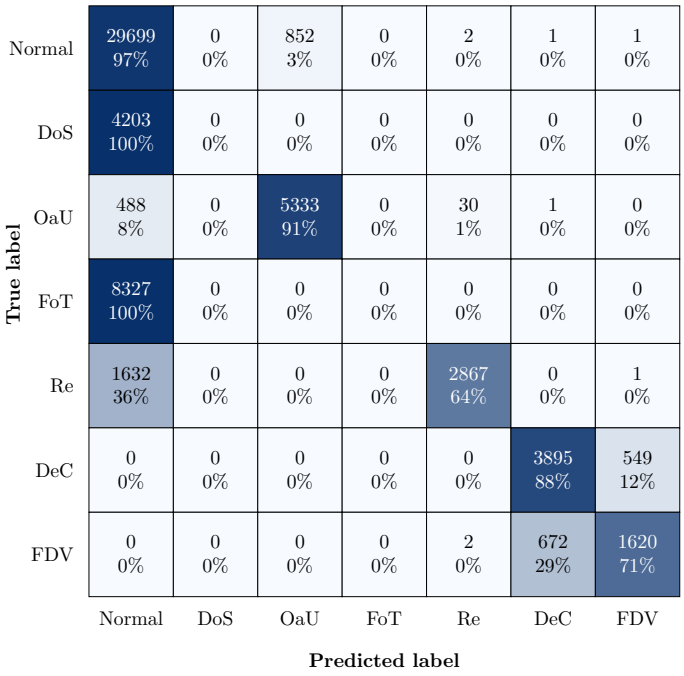
4.3.4.5 Processing Time

Figure 4.9 describes the processing time of two ENs with the same Co-CNN-5 model. We can observe in Figure 4.9 that when the number of transactions increases, the processing time of both ENs also linearly increases. However, there is a different capacity between the two ENs. In detail, while EN-5 can process about 1,100 transactions per second, the number of transactions that EN-1 can process is around 2,150 transactions per second. This is because of the different types of computer configuration between the two ENs described in section 4.3.1. However, in the mainnet of the Ethereum system, the maximum recorded number of transactions is 93.01 per second [89]. Therefore, the capacity of our proposed system can be well-adapted to detect attacks on the mainnet Ethereum system.

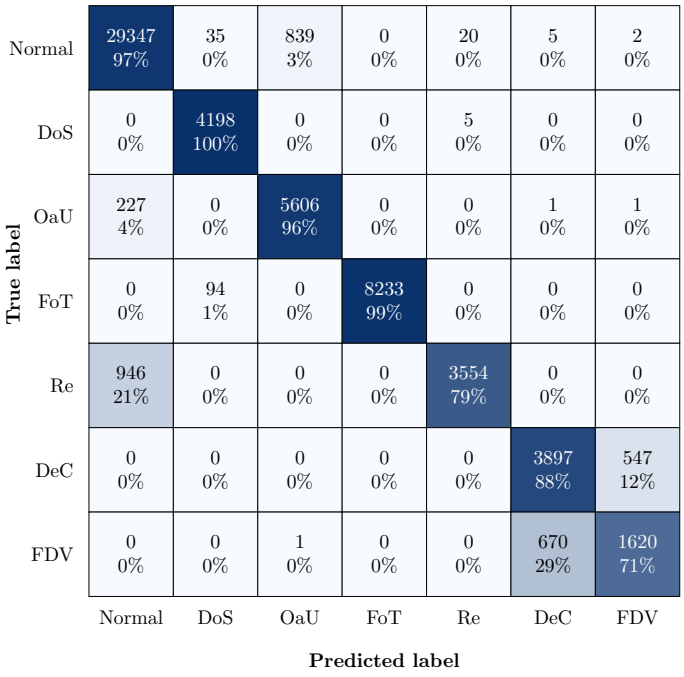
4.4 Conclusion

In this chapter, we have developed a collaborative learning model that could efficiently detect malicious attacks in transactions and smart contracts in a blockchain network. To do this, we have implemented a private Ethereum network in our laboratory. We have then performed attacks in transactions and SCs of that network for analysis. Next, we have analyzed the transaction data and extracted the important features (i.e., Bytecode and value) to build the dataset. After that, we have

converted the dataset into the image format to train and evaluate the performance of our proposed model. In our proposed model, a learning node can detect attacks in transactions and SCs of a blockchain network. It then can receive and aggregate learning knowledge (i.e., trained models) from other learning nodes to improve the accuracy of detection. As a result, our proposed model safeguards the privacy of the local data at the learning nodes by ensuring that this information is not exposed over the network. Both simulation results and real-time experimental results showed the efficiency of our proposed model in detecting attacks.

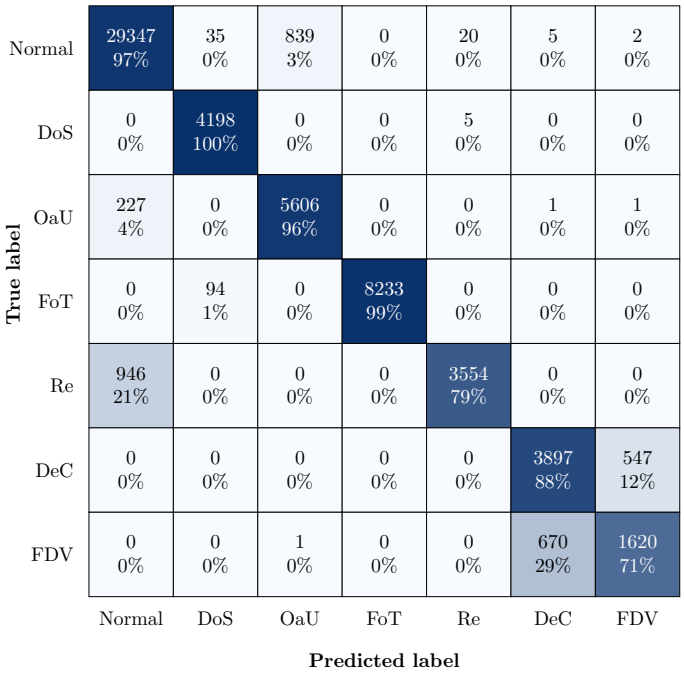


(a)

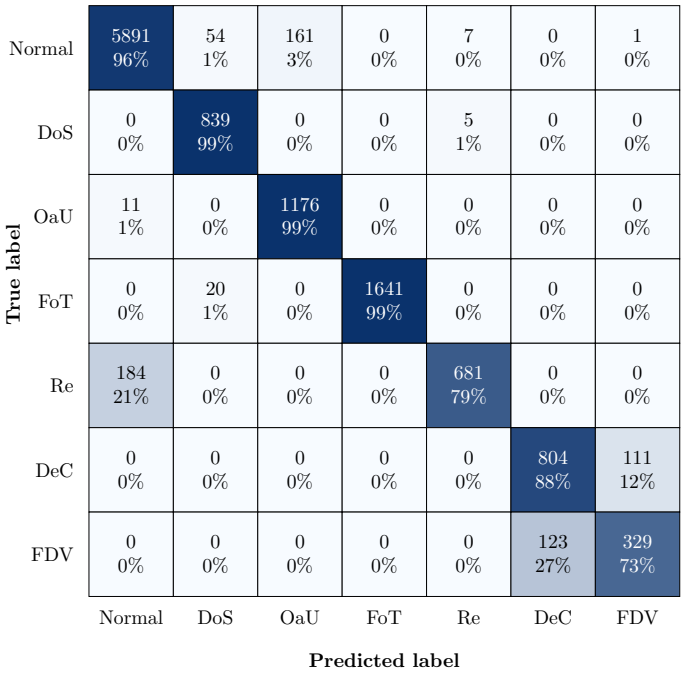


(b)

Figure 4.5: The detection results of the models w/ and w/o-V feature. (a) Centralized-CNN w/o-V. (b) Centralized-CNN w/-V.

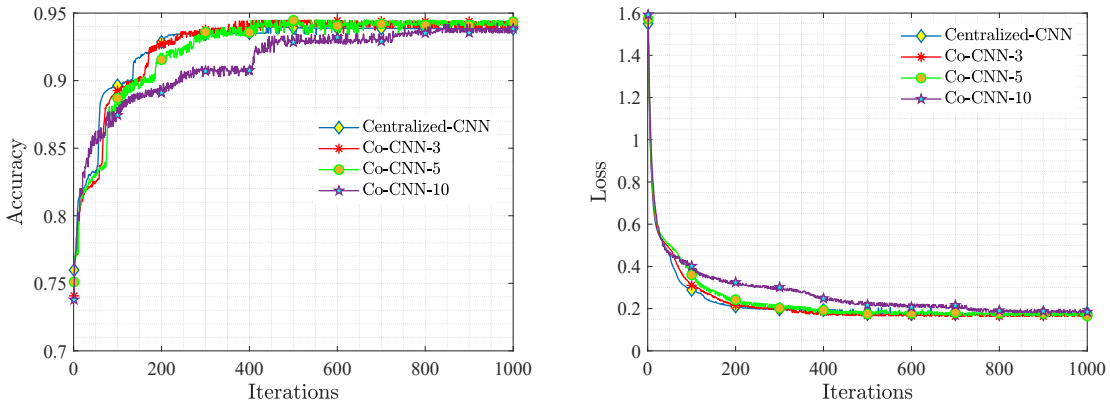


(a)



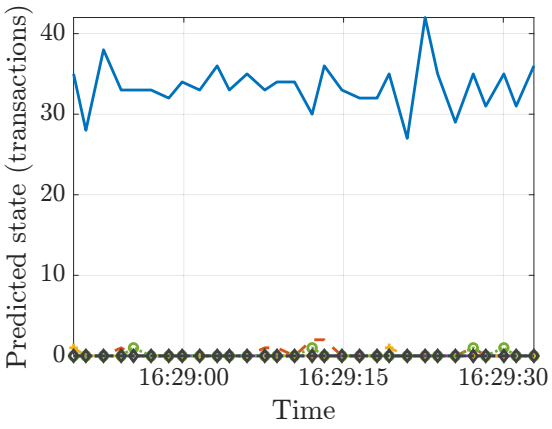
(b)

Figure 4.6: The detection results of Centralized-CNN and Co-CNN-5 models. (a) Centralized-CNN w/-V. (b) Co-CNN-5 w/-V.

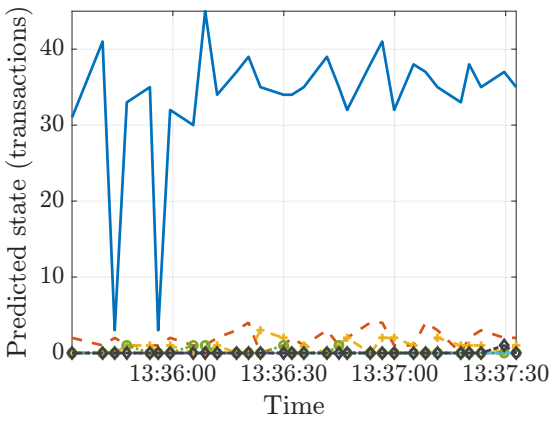


(a) (b)

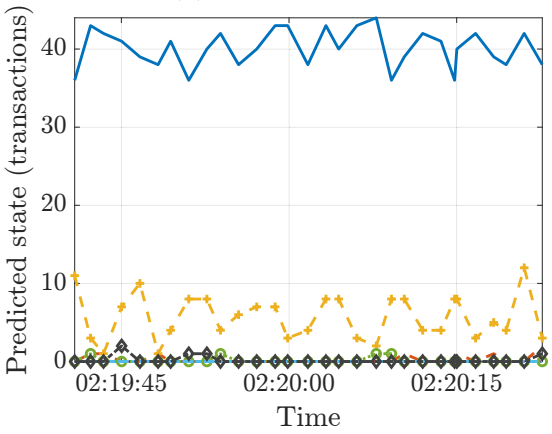
Figure 4.7: The convergence of accuracy and loss over iterations: (a) The accuracy over interactions, and (b) The loss over iterations.



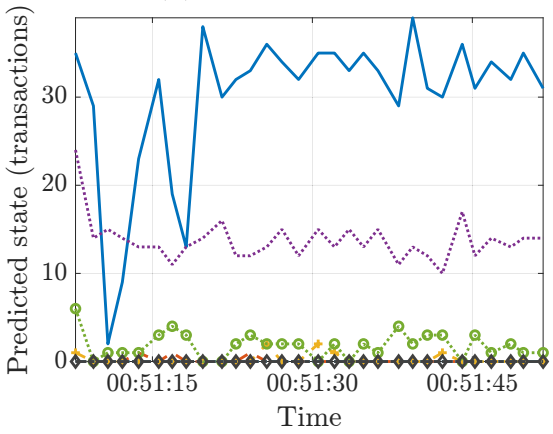
(a) Normal state



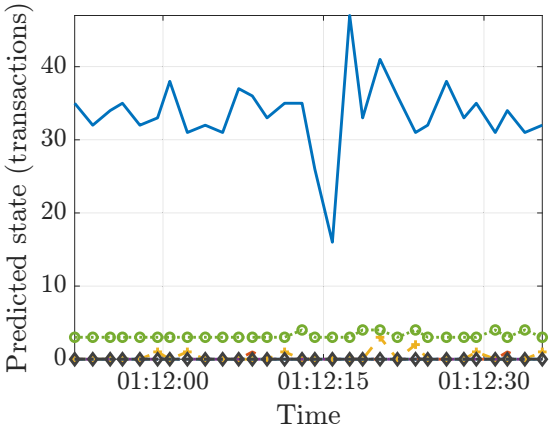
(b) Re attack state



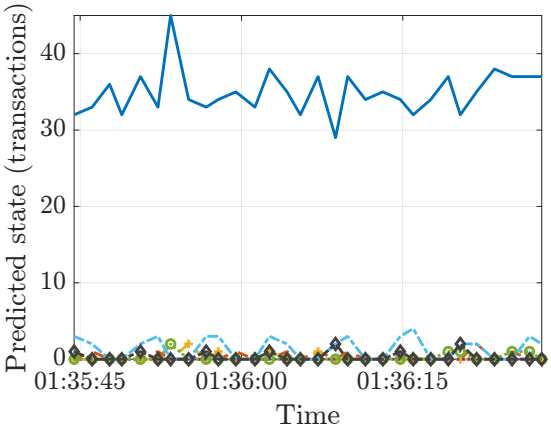
(c) OaU attack state



′d) FoT attack state



(e) DoS attack state



(f) DeC attack state

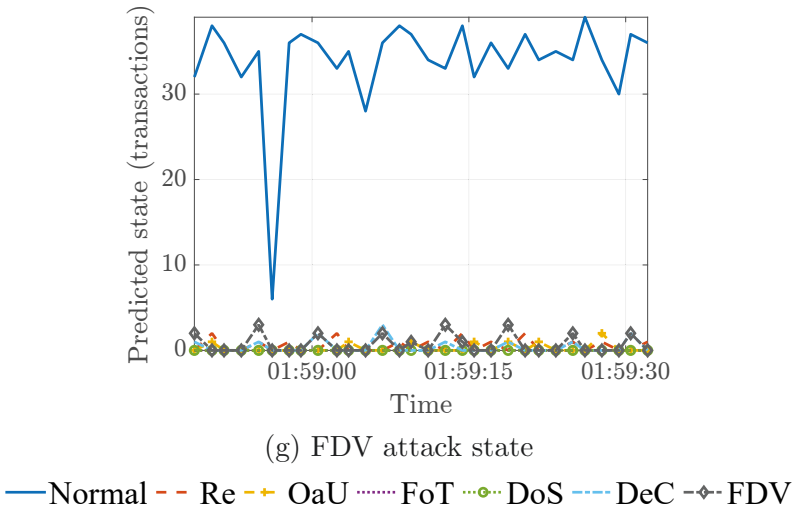


Figure 4.8: Real-time cyberattack detection: proposed Co-CNN-5 model in Ethereum node 1.

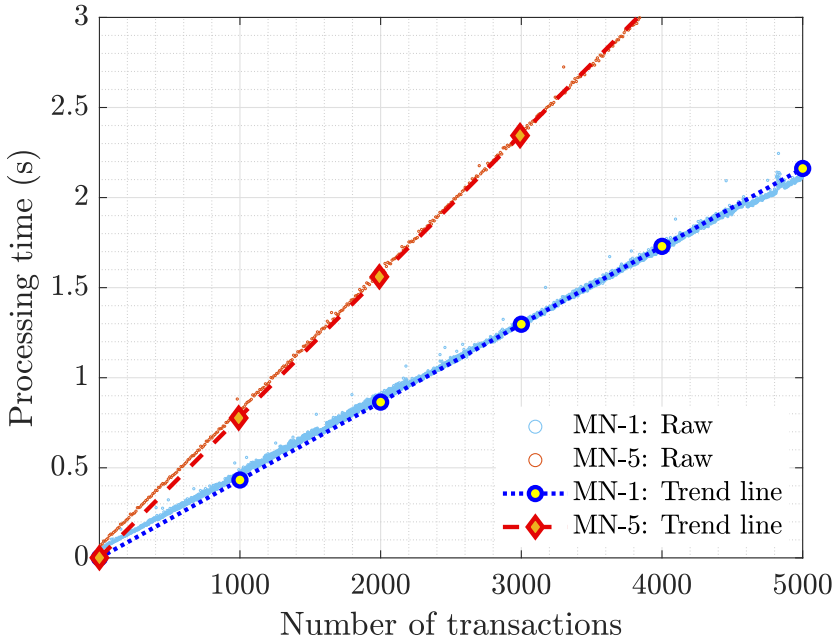


Figure 4.9: The processing time of proposed Co-CNN-5 model in two computer configurations.

Chapter 5

Conclusions and Potential Research Directions

5.1 Conclusions

In this thesis, we have presented our works in developing cyberattack detection models based on FL and TL for decentralized networks. In each chapter, we have deeply analyzed challenges and proposed a collaborative learning model to detect cyberattacks in IoT or Blockchain networks.

In Chapter 2, we have proposed a novel collaborative learning framework to address the limitations of current ML-based cyberattack detection systems in IoT networks. By extracting and transferring knowledge from a network with abundant labeled data (source network), the intrusion detection performance of the target network could be significantly improved (even if the target has very few labeled data). More importantly, unlike most of the current works in this area, our proposed framework could enable the source network to transfer the knowledge to the target network even when they are different data structures, e.g., different features. The experimental results then showed that the accuracy of prediction of our proposed framework was significantly improved in comparison with the state-of-the-art UDL model. In addition, the convergence of the proposed collaborative learning model was also analyzed with various cybersecurity datasets. In future work, we can consider using other effective TL techniques to make TL processes more stable and achieve better performance, especially when the amount of mutual information is very limited.

In Chapter 3, we have proposed a novel collaborative learning framework for a

cyberattack detection system in a blockchain network. First, we have implemented a private blockchain network in our laboratory. This blockchain network was used to (1) generate data (both normal and attack data) to serve the proposed learning models and (2) validate the performance of our proposed learning framework in real-time experiments. We then build an effective BC-ID tool to collect data in the blockchain network. This tool can extract features from the collected network traffic data, filter attack samples in network traffic, and exactly label them in a real-time manner. After that, we have proposed a highly-effective learning model that allows to be effectively deployed in the blockchain network. This learning model allows nodes in the blockchain can be actively involved in the detection process by collecting data, learning knowledge from their data, and then exchanging knowledge together to improve the attack detection ability. In this way, we can not only avoid problems of conventional centralized learning (e.g., congestion and single point of failure) but also protect the blockchain network right at the edge. Both simulation and real-time experimental results then have clearly shown the efficiency of our proposed framework. In the future, we plan to continue developing this dataset with other emerging types of attacks and develop more effective methods to protect blockchain networks.

In Chapter 4, we have developed a collaborative learning model that could efficiently detect malicious attacks in transactions and smart contracts in a blockchain network. To do this, we have implemented a private Ethereum network in our laboratory. We have then performed attacks in transactions and SCs of that network for analysis. Next, we have analyzed the transaction data and extracted the important features (i.e., Bytecode and value) to build the dataset. After that, we have converted the dataset into the image format to train and evaluate the performance of our proposed model. In our proposed model, a learning node can detect attacks in transactions and SCs of a blockchain network. It then can receive and aggregate

learning knowledge (i.e., trained models) from other learning nodes to improve the accuracy of detection. As a result, our proposed model safeguards the privacy of the local data at the learning nodes by ensuring that this information is not exposed over the network. Both simulation results and real-time experimental results showed the efficiency of our proposed model in detecting attacks.

5.2 Potential Future Research Directions

Cyberattack detection faces numerous challenges with the increasing number of new types of attacks as well as emerging decentralized systems such as autonomous vehicle systems, satellite networks, and Metaverse.

5.2.1 FL for cyberattack detection in autonomous vehicle systems

The prospect of research in FL for cyberattack detection in autonomous vehicle systems is highly promising, particularly as the automotive industry strides toward an increasingly connected and autonomous future. One key avenue of exploration lies in enhancing the scalability and efficiency of FL algorithms to accommodate the complex and dynamic nature of cyber threats targeting autonomous vehicles. Researchers can focus on developing robust FL models that can adapt to diverse environments and evolving attack strategies, ensuring the resilience of autonomous systems against adversarial manipulations. Moreover, the integration of advanced anomaly detection techniques within the FL framework will be pivotal in identifying subtle deviations from normal behavior, thereby fortifying the cyber resilience of autonomous vehicle networks. As the automotive landscape embraces greater connectivity, exploring privacy-preserving FL approaches will be crucial to address concerns surrounding data security and user privacy. Additionally, interdisciplinary collaborations between experts in machine learning, cybersecurity, and automotive engineering will play a pivotal role in shaping the future of FL research, fostering

innovations that enhance the reliability and security of autonomous vehicles in the face of emerging cyber threats.

5.2.2 FL for cyberattack detection in satellite systems

The future of research in FL for cyberattack detection in satellite systems holds tremendous potential as the space industry continues to advance. As satellites become integral to global communication, navigation, and Earth observation, the need for robust cybersecurity measures is paramount. Future investigations are likely to focus on optimizing FL algorithms to cater to the unique challenges posed by satellite networks, including limited bandwidth, high latency, and intermittent connectivity. Researchers will explore ways to enhance the efficiency of FL models in detecting and mitigating cyber threats such as jamming, spoofing, and unauthorized access to satellite systems. Additionally, there will be a concerted effort to develop FL frameworks that are resilient to adversarial attacks and can adapt to the evolving tactics of cyber adversaries. Interdisciplinary collaborations between experts in machine learning, satellite technology, and cybersecurity will play a crucial role in shaping the future of FL research for satellite systems, ensuring the security and reliability of these critical space assets in the face of emerging cyber challenges.

5.2.3 FL for cyberattack detection in Metaverse

In the rapidly evolving domain of cybersecurity, the concept of FL holds immense potential, particularly in the context of the Metaverse. The Metaverse, a collective virtual shared space created by the convergence of virtually enhanced physical and digital reality, is inherently complex and decentralized. This complexity presents unique challenges and opportunities for cyberattack detection. By leveraging decentralized data sources across various nodes within the Metaverse without compromising data privacy, FL can provide a robust framework for detecting and mitigating cyber threats. This approach not only protects the privacy and autonomy of individ-

ual data sources but also harnesses their collective intelligence to improve detection algorithms. The exploration will focus on developing adaptable, efficient, and scalable FL models that can effectively identify and counteract evolving cyber threats in the Metaverse, potentially setting new benchmarks in cybersecurity for virtual ecosystems.

Bibliography

[1] Y. Keshet, "Half of the malware detected in 2019 was classified as zero-day threats, making it the most common malware to date," Mar. 2020. [Online]. Available: https://www.cynet.com/blog/half-of-the-malware-detected-in-2019-was-classified-as-zero-day-threats-making-it-the-most-common-malware-to-date/

[2] S. Morgan, "Global ransomware damage costs predicted to hit $11.5 billion by 2019," Mar. 2021. [Online]. Available: https://cybersecurityventures.com/ransomware-damage-report-2017-part-2/

[3] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, and M. H. Rehmani, "Applications of blockchains in the Internet of Things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1676–1717, Dec. 2018.

[4] J. Xie, H. Tang, T. Huang, F. R. Yu, R. Xie, J. Liu, and Y. Liu, "A survey of blockchain technology applied to smart cities: Research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2794–2830, Feb. 2019.

[5] S. Biswas, K. Sharif, F. Li, Z. Latif, S. S. Kanhere, and S. P. Mohanty, "Interoperability and synchronization management of blockchain-based decentralized e-health systems," *IEEE Transactions on Engineering Management*, vol. 67, no. 4, pp. 1363–1376, June 2020.

[6] Y. Yuan and F.-Y. Wang, "Blockchain and cryptocurrencies: Model, tech-

niques, and applications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 9, pp. 1421–1428, July 2018.

[7] "The 10 Biggest Crypto Exchange Hacks In History," Accessed: Apr. 11, 2024. [Online]. Available: https://crystalblockchain.com/articles/the-10-biggest-crypto-exchange-hacks-in-history

[8] "North Korean Hackers Have Prolific Year as Their Unlaundered Cryptocurrency Holdings Reach All-time High," Accessed: Apr. 11, 2024. [Online]. Available: https://blog.chainalysis.com/reports/north-korean-hackers-have-prolific-year-as-their-total-unlaundered-cryptocurrency-holdings-reach-all-time-high

[9] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, "Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities," *IEEE Access*, vol. 7, pp. 85 727–85 745, Jun. 2019.

[10] K. Salah, N. Nizamuddin, R. Jayaraman, and M. Omar, "Blockchain-based soybean traceability in agricultural supply chain," *IEEE Access*, vol. 7, pp. 73 295–73 305, May 2019.

[11] T. V. Khoa, D. H. Son, D. T. Hoang, N. L. Trung, T. T. T. Quynh, D. N. Nguyen, N. V. Ha, and E. Dutkiewicz, "Collaborative learning for cyberattack detection in blockchain networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Apr. 2024.

[12] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, Apr. 2020.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[14] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 70–82, Apr. 2020.

[15] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017.

[16] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," *Neurocomputing*, vol. 184, pp. 232–242, Apr. 2016.

[17] L. Vu, Q. U. Nguyen, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, "Deep transfer learning for IoT attack detection," *IEEE Access*, vol. 8, pp. 107 335–107 344, June 2020.

[18] T. V. Khoa, Y. M. Saputra, D. T. Hoang, N. L. Trung, D. Nguyen, N. V. Ha, and E. Dutkiewicz, "Collaborative learning model for cyberattack detection systems in iot industry 4.0," in *2020 IEEE Wireless Communications and Networking Conference*, May 2020, pp. 1–6.

[19] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, July 2006.

[20] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, Aug. 2017.

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[22] T. V. Khoa, D. H. Son, C.-H. Nguyen, D. T. Hoang, D. N. Nguyen, N. L. Trung, T. T. T. Quynh, T.-M. Hoang, N. V. Ha, and E. Dutkiewicz, "Securing

blockchain systems: A novel collaborative learning framework to detect attacks in transactions and smart contracts," *arXiv preprint arXiv:2308.15804*, Aug. 2023.

[23] Y. M. Saputra, D. Nguyen, H. T. Dinh, Q.-V. Pham, E. Dutkiewicz, and W.-J. Hwang, "Federated learning framework with straggling mitigation and privacy-awareness for ai-based mobile application services," *IEEE Transactions on Mobile Computing*, May 2022.

[24] 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*, Apr. 2017, pp. 1273–1282.

[25] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, Jan. 2019.

[26] S. Niu, Y. Liu, J. Wang, and H. Song, "A decade survey of transfer learning (2010–2020)," *IEEE Transactions on Artificial Intelligence*, vol. 1, no. 2, pp. 151–166, Oct. 2020.

[27] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct. 2009.

[28] C. T. Nguyen, N. Van Huynh, N. H. Chu, Y. M. Saputra, D. T. Hoang, D. N. Nguyen, Q.-V. Pham, D. Niyato, E. Dutkiewicz, and W.-J. Hwang, "Transfer learning for wireless networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 110, no. 8, pp. 1073–1115, June 2022.

[29] K. E. Mwangi, S. Masupe, and J. Mandu, "Transfer learning for internet of things malware analysis," in *International Conference on Information, Communication and Computing Technology*, Oct. 2019, pp. 198–208.

[30] Y. Fan, Y. Li, M. Zhan, H. Cui, and Y. Zhang, "Iotdefender: A federated transfer learning intrusion detection framework for 5g iot," in *2020 IEEE 14th International Conference on Big Data Science and Engineering*, Dec. 2020, pp. 88–95.

[31] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, Apr. 2019.

[32] 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, July 2018.

[33] N. Moustafa, "A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets," *Sustainable Cities and Society*, vol. 72, p. 102994, Sep. 2021.

[34] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali, and M. Guizani, "A survey of machine and deep learning methods for internet of things (iot) security," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1646–1685, Apr. 2020.

[35] A. Abeshu and N. Chilamkurti, "Deep learning: The frontier for distributed attack detection in fog-to-things computing," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 169–175, Feb. 2018.

[36] 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*, July 2019, pp. 756–767.

[37] V. Mothukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, "Federated-learning-based anomaly detection for iot security attacks," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2545–2554, Feb. 2021.

[38] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, Apr. 2021.

[39] S. T. Mehedi, A. Anwar, Z. Rahman, K. Ahmed, and R. Islam, "Dependable intrusion detection system for iot: A deep transfer learning based approach," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 1006–1017, Apr. 2022.

[40] S. Yılmaz, E. Aydogan, and S. Sen, "A transfer learning approach for securing resource-constrained iot devices," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4405–4418, July 2021.

[41] T. Wen and R. Keyes, "Time series anomaly detection using convolutional neural networks and transfer learning," *arXiv preprint arXiv:1905.13628*, May 2019.

[42] C. Zhao, Z. Cai, M. Huang, M. Shi, X. Du, and M. Guizani, "The identification of secular variation in IoT based on transfer learning," in *2018 International Conference on Computing, Networking and Communications*, Mar. 2018, pp. 878–882.

[43] Y. Sharaf-Dabbagh and W. Saad, "Transfer learning for device fingerprinting with application to cognitive radio networks," in *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications*, Aug. 2015, pp. 2138–2142.

[44] M. U. Hassan, M. H. Rehmani, and J. Chen, "Anomaly detection in blockchain networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 289–318, Sep. 2022.

[45] P. Kumar, R. Kumar, G. Gupta, and R. Tripathi, "A distributed framework for detecting DDoS attacks in smart contract-based Blockchain-IoT systems by leveraging fog computing," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 6, pp. 1–31, June 2021.

[46] O. Alkadi, N. Moustafa, B. Turnbull, and K.-K. R. Choo, "A deep blockchain framework-enabled collaborative intrusion detection for protecting IoT and cloud networks," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9463–9472, June 2020.

[47] J. Kim, M. Nakashima, W. Fan, S. Wuthier, X. Zhou, I. Kim, and S.-Y. Chang, "Anomaly detection based on traffic monitoring for secure blockchain networking," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Sydney, Australia, May 2021, pp. 1–9.

[48] Z. Liu and X. Yin, "LSTM-CGAN: towards generating low-rate DDoS adversarial samples for blockchain-based wireless network detection models," *IEEE Access*, vol. 9, pp. 22 616–22 625, Feb. 2021.

[49] W. Cao, Y. Huang, D. Li, F. Yang, X. Jiang, and J. Yang, "A blockchain based link-flooding attack detection scheme," in *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 4, Chongqing, China, June 2021, pp. 1665–1669.

[50] P. Ramanan, D. Li, and N. Gebraeel, "Blockchain-based decentralized replay attack detection for large-scale power systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Aug. 2021.

[51] B. Hu, C. Zhou, Y.-C. Tian, Y. Qin, and X. Junping, "A collaborative intrusion detection approach using blockchain for multimicrogrid systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 8, pp. 1720–1730, Apr. 2019.

[52] P. Qian, Z. Liu, Q. He, R. Zimmermann, and X. Wang, "Towards automated reentrancy detection for smart contracts based on sequential models," *IEEE Access*, vol. 8, pp. 19 685–19 695, Jan. 2020.

[53] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "Contractward: Automated vulnerability detection models for ethereum smart contracts," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1133–1144, Jan. 2020.

[54] Q.-B. Nguyen, A.-Q. Nguyen, V.-H. Nguyen, T. Nguyen-Le, and K. Nguyen-An, "Detect abnormal behaviours in ethereum smart contracts using attack vectors," in *International Conference on Future Data and Security Engineering*, Nha Trang City, Vietnam, Nov. 2019, pp. 485–505.

[55] J. Huang, S. Han, W. You, W. Shi, B. Liang, J. Wu, and Y. Wu, "Hunting vulnerable smart contracts via graph embedding based bytecode matching," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2144–2156, Jan. 2021.

[56] N. Ivanov, Q. Yan, and A. Kompalli, "Txt: Real-time transaction encapsulation for Ethereum smart contracts," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1141–1155, Jan. 2023.

[57] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "Defectchecker: Automated smart contract defect detection by analyzing evm bytecode," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2189–2207, Jan. 2021.

[58] M. A. Ferrag, O. Friha, L. Maglaras, H. Janicke, and L. Shu, "Federated deep learning for cyber security in the internet of things: Concepts, applications, and experimental analysis," *IEEE Access*, vol. 9, pp. 138 509–138 542, Oct. 2021.

[59] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, Feb. 2018.

[60] "KDD dataset," http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[61] "NSL-KDD dataset," https://www.unb.ca/cic/datasets/nsl.html.

[62] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference*, Nov. 2015, pp. 1–6.

[63] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, June 2006.

[64] D. M. Powers, "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation," *Journal of Machine Learning Technologies*, pp. 37–63, Oct. 2011.

[65] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008, Accessed: Apr. 11, 2024. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[66] D. H. Son, T. T. T. Quynh, T. V. Khoa, D. T. Hoang, N. L. Trung, N. V. Ha, D. Niyato, D. N. Nguyen, and E. Dutkiewicz, "An effective framework of private ethereum blockchain networks for smart grid," in *2021 International Conference on Advanced Technologies for Communications (ATC)*. IEEE, Oct. 2021, pp. 312–317.

[67] "kdd99_feature_extractor," AI-IDS, Accessed: Apr. 11, 2024. [Online]. Available: https://github.com/AI-IDS/kdd99_feature_extractor

[68] "Wireshark Network Analysis," Accessed: Apr. 11, 2024. [Online]. Available: https://www.wireshark.org

[69] 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.

[70] Ethereum, "Official Go implementation of the Ethereum protocol," Accessed: Apr. 11, 2024. [Online]. Available: https://github.com/ethereum/go-ethereum

[71] M. Oance, "Ethereum Network Stats," Accessed: Apr. 11, 2024. [Online]. Available: https://github.com/cubedro/eth-netstats

[72] T. Neudecker and H. Hartenstein, "Network layer aspects of permissionless blockchains," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 838–857, Sep. 2019.

[73] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and D. Mohaisen, "Exploring the attack surface of blockchain: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1977–2008, Mar. 2020.

[74] "Bitfinex restored after DDoS attack," Accessed: Apr. 11, 2024. [Online]. Available: https://malware.news/t/bitfinex-restored-after-ddos-attack/16933

[75] J. Otávio Chervinski, D. Kreutz, and J. Yu, "Analysis of transaction flooding attacks against Monero," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Sydney, Australia, May 2021, pp. 1–8.

[76] X. Wang, X. Zha, G. Yu, W. Ni, R. P. Liu, Y. J. Guo, X. Niu, and K. Zheng, "Attack and defence of ethereum remote apis," in *2018 IEEE Globecom Workshops (GC Wkshps)*, Abu Dhabi, United Arab Emirates, Dec. 2018, pp. 1–6.

[77] "KDD Cup 1999 Data," The Fifth International Conference on Knowledge Discovery and Data Mining, Accessed: Apr. 11, 2024. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[78] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, Nov. 2008.

[79] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Computing Surveys*, vol. 53, no. 3, pp. 1–43, May 2021.

[80] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and D. Mohaisen, "Exploring the attack surface of blockchain: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1977–2008, Mar. 2020.

[81] L. Hollander, "Evm bytecode decompiler," Accessed: Apr. 11, 2024. [Online]. Available: https://www.npmjs.com/package/evm

[82] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations*, San Diego, CA, USA, May 2015, pp. 1–15.

[83] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[84] Prysmatic Labs, "Prysm: An Ethereum Consensus Implementation

Written in Go," Accessed: Apr. 11, 2024. [Online]. Available: https://github.com/prysmaticlabs/prysm/tree/v3.2.0

[85] A. Said, M. U. Janjua, S.-U. Hassan, Z. Muzammal, T. Saleem, T. Thaipisutikul, S. Tuarob, and R. Nawaz, "Detailed analysis of ethereum network on transaction behavior, community structure and link prediction," *PeerJ Computer Science*, vol. 7, pp. 1–26, Dec. 2021.

[86] OpenZeppelin, "A library for secure smart contract development," Accessed: Apr. 11, 2024. [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts

[87] SmartContractSecurity, "SWC Registry - Smart Contract Weakness Classification and Test Cases," Accessed: Apr. 11, 2024. [Online]. Available: https://swcregistry.io

[88] E. Banisadr, "How $800k Evaporated from the PoWH Coin Ponzi Scheme Overnight," Accessed: Apr. 11, 2024. [Online]. Available: https://medium.com/@ebanisadr/how-800k-evaporated-from-the-powh-coin-ponzi-scheme-overnight-1b025c33b530

[89] Etherscan, "Ethereum daily transactions chart," Accessed: Apr. 11, 2024. [Online]. Available: https://etherscan.io/chart/tx