# Securing Blockchain Systems: A Novel Collaborative Learning Framework to Detect Attacks in Transactions and Smart Contracts

Tran Viet Khoa, Do Hai Son, Chi-Hieu Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Nguyen Linh Trung, Tran Thi Thuy Quynh, Trong-Minh Hoang, Nguyen Viet Ha, Eryk Dutkiewicz, and Mohammad Abu Alsheikh.

***Abstract*—With the escalating prevalence of malicious activities exploiting vulnerabilities in blockchain systems, there is an urgent requirement for robust attack detection mechanisms. To address this challenge, this paper presents a novel collaborative learning framework designed to detect attacks in blockchain transactions and smart contracts by analyzing transaction features. Our framework exhibits the capability to classify various types of blockchain attacks, including intricate attacks at the machine code level (e.g., injecting malicious codes to withdraw coins from users unlawfully), which typically necessitate significant time and security expertise to detect. To achieve that, the proposed framework incorporates a unique tool that transforms transaction features into visual representations, facilitating efficient analysis and classification of low-level machine codes. Furthermore, we propose a customized collaborative learning model to enable real-time detection of diverse attack types at distributed mining nodes. In order to create a comprehensive dataset, we deploy a pilot system based on a private Ethereum network and conduct multiple attack scenarios. To the best of our knowledge, our dataset is the most comprehensive and diverse collection of transactions and smart contracts synthesized in a laboratory for cyberattack detection in blockchain systems. Our framework achieves a detection accuracy of approximately 94% through extensive simulations and real-time experiments with a throughput of over 2,150 transactions per second. These compelling results validate the efficacy of our framework and showcase its adaptability in addressing real-world cyberattack scenarios.**

***Index Terms*—Cybersecurity, cyberattack detection, deep learning, blockchain, smart contract.**

## I. Introduction

BLOCKCHAIN technology has been rapidly being developed with many applications in recent years. This technology was initially developed with a well-known digital currency application named Bitcoin. After that, many potential applications using this technology have been developed beyond cryptocurrency. The tremendous development of this technology is from the fact that it provides a new approach to data sharing and storage without the need for any third party (e.g., bank and government). Blockchain is a decentralized environment in which transactions and smart contracts can be recorded and executed in a secure and transparent manner. It is challenging to manipulate transactions once they are put into the blocks. Thus, blockchain technology protects data integrity, and its applications have been being widely developed in various fields of industry such as smart manufacturing, supply chain management, food industry, smart grid, healthcare, and Internet of Things [1].

Smart Contracts (SCs) are solely programs in blockchain systems (e.g., Ethereum and Solana). Smart contracts define and enforce a set of rules for users via using codes. They also facilitate users' interactions by allowing them to send transactions to execute defined functions. By default, smart contracts and their interactions are irreversible [2]. However, in practical scenarios, attackers can inject malicious codes into smart contracts and transactions to attack a blockchain system for specific purposes. For instance, smart contracts exhibit various vulnerabilities [3], which attackers can exploit to engage in injurious purposes, including unauthorized coin withdrawals from other users' pockets and taking control of the system [4]. Specifically, in 2016, a smart contract (SC) named Decentralized Autonomous Organization (DAO) was a victim of a re-entrancy attack. At that time, this SC held $150 million in the Ethereum network, and this attack led to a hardfork of Ethereum that created Ethereum Classic (ETC) [3]. In addition, the 4Chan group created an SC named Proof of Weak Hands Coin (PoWHC) on the Ethereum system. However, this SC witnessed an underflow attack that caused a loss of 866 ETH (i.e., Ethereum coins) [5]. Although most of the attacks in blockchain systems happened in the finance sector, many blockchain-based applications have been developing in different sectors such as healthcare, supply chain, and food industry [1]. Therefore, securing blockchain systems against cyber threats has become an imperative necessity.

There are a number of challenges to detect and prevent attacks in transactions and SCs. The first challenge is the lack of a dataset synthesized in the laboratory for various kinds of attacks on transactions and SCs in a blockchain system. In recent research (e.g., [6] and [7]), the authors use datasets from the public blockchain network and label data using the attack records history. When using this method to label attack data, it is assumed that the benign data does not include the insight attacks. Therefore, generating data, which has "clean" samples (i.e., transactions between users

T. V. Khoa, C. H. Nguyen, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz are with the School of Electrical and Data Engineering, University of Technology Sydney, Sydney, NSW 2007, Australia (e-mail: {khoa.v.tran, hieu.c.nguyen}@student.uts.edu.au, {hoang.dinh, diep.nguyen, eryk.dutkiewicz}@uts.edu.au).

D. H. Son is with the VNU Information Technology Institute, Hanoi, Vietnam (e-mail: dohaison1998@vnu.edu.vn).

N. L. Trung, T. T. T. Quynh, and N. V. Ha are with the Advanced Institute of Engineering and Technology (AVITECH), University of Engineering and Technology, Vietnam National University, Hanoi, Vietnam (e-mail: {linhtrung, quynhttt, hanv}@vnu.edu.vn).

Mohammad Abu Alsheikh is with the University of Canberra, Australia (e-mail: mohammad.abualsheikh@canberra.edu.au).

Trong-Minh Hoang is with the Posts and Telecommunications Institute of Technology, Vietnam (e-mail: hoangtrongminh@ptit.edu.vn).

without any malicious behaviour) of normal behavior and attacks in transactions and SCs, is urgently needed. However, a blockchain system in the Mainnet has large and diverse types of data. Thus, a synthesized dataset from the laboratory needs to be diverse and similar to reality. The second challenge is to understand and analyze the content of Bytecode, the compiled form of an SC's source code. It is worth noting that the main functions of the transactions and smart contracts are encoded into the Bytecode, which is represented by a series of hexadecimal numbers, to be implemented in a blockchain system [4]. It is crucial for a real-time attack detection system in analyzing the content of Bytecode to detect attacks in a blockchain system [6]. There are two approaches to analyze the Bytecode, i.e., using the source code of SCs for comparison and analyzing the Bytecode. Unfortunately, only 1% source codes of SCs are open [6], and analyzing Bytecode without the corresponding source code of smart contracts and transactions can be unreliable and time-consuming [6]. The third challenge is that most of the current attack detection models are centralized. Thus, they need to gather all data (i.e., transactions together with their labels, e.g., attack or normal) into a centralized model to perform training and testing. However, blockchain systems are decentralized environments so it is challenging to collect data from all mining nodes (MNs) to perform training at the centralized server. In addition, if we transfer data from all MNs to the centralized server for processing (e.g., training and testing), data privacy can be compromised.

Given the above, in this paper, 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). These attacks were recorded as occurring in the real world and resulted in serious consequences for the Ethereum system. To address the first challenge, we collect all the transactions in MNs to build a dataset, called **Blockchain Transaction Attacks Dataset (BTAD)**[1]. To the best of our knowledge, 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 second challenge of Bytecode analysis, 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 an image form. This tool calls the transaction using a transaction hash (i.e., a feature of the transaction) and then extracts key fields like Bytecode and value from the transactions. 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 framework 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 highly-effective 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 mining 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. Our major contributions can be summarized as follows:

- We implement a blockchain system and perform experiments to collect the BTAD 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 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 mining nodes to detect attacks in transactions and SCs for a blockchain network. In our framework, the mining 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 mining 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 an accuracy of 94% without exposing the mining node's local dataset over the network.
- We perform both simulations and real-time testing 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.

## II. Related work

There are several works trying to deal with attacks on transactions and SCs in blockchain networks. In [8], the authors propose to convert the source codes of smart contracts into vectors. They then use bidirectional long-short-term memory to identify abnormal patterns of vectors to detect re-entrancy attacks. The simulation results show that their proposed model can achieve 88.26% F1-Score and 88.47% accuracy in detecting re-entrancy attacks. In [9], the authors propose to detect the vulnerabilities inside SCs. To do this,

[1]https://avitech-vnu.github.io/BTAD

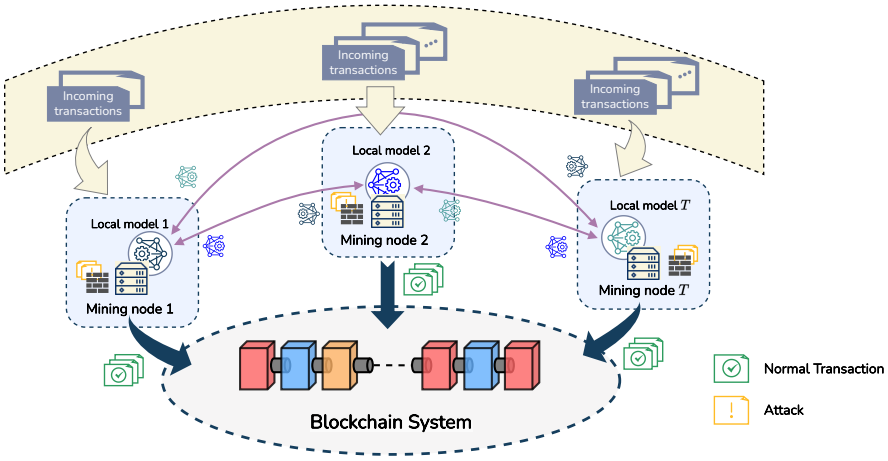



Fig. 1: The system model of our proposed framework. While receiving transactions, our framework will perform preprocessing to extract important information. After that, our collaborative learning will perform the attack detection process to detect network normal behaviour or a type of attack.

they use feature extraction to analyze the Bytecode of SCs. In this paper, the authors use various types of machine learning models to detect 6 types of vulnerabilities with an F1-score of up to 97%. Even though the methods in [8], [9] can detect some types of attacks, they need to use source code 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 MNs for the mining process. From the MN 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 deep learning approaches, in [10], the authors study the Bytecode. They propose to use the attack vector method to directly analyze the Bytecode. This approach can effectively detect some specific attacks by using pre-defined vectors. However, this method is difficult to extend to various types of new attacks. In addition, even though the attack detection ability can achieve up to 100% in some types of attacks (e.g., re-entrancy, delegatecall, overflow, etc), the authors only test this method in a small scale of data (about 100 samples). In [6] the authors propose to use Graph embedding to analyze Bytecode. To do this, the authors convert the Bytecode of SC into vectors and then compare the similarities between the vectors of SC to detect the insight attacks of SC. The experimental results show that this method can achieve a precision of up to 91.95% in detecting attacks. Both [10] and [6] have to use source code to analyze the bytecode. In [11], the authors introduce 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 point out that further Bytecode analysis can reduce this portion. In addition, in [7], the authors propose DefectChecker to analyze vulnerabilities in SCs. This framework uses symbolic execution to analyze Bytecode without the need for source codes. This framework can detect eight types of vulnerabilities in SCs and get an F1-score of 88%. Unlike all the above works and others in the literature, in this paper, we introduce an innovative ML-based framework to analyze Bytecode directly to detect attacks inside 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 effectively detect new types of attacks. 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 MNs are distributed worldwide. Thus, gathering all blockchain data to perform training and testing is impractical.

## III. Blockchain System: Fundamental and Proposed Collaborative Learning Framework

### A. Blockchain

Blockchain technology is a decentralized method to store and manage data. In a blockchain system, each MN can be used to store and process data. When a Mining Node (MN) receives transactions, it typically groups them into a block as a part of the mining process. However, it is worth noting that the consensus mechanism is responsible for managing the rules of the mining process in a blockchain network. There are various types of consensus mechanisms being used in blockchain networks [12]. For example, Ethereum 2.0 uses Proof-of-Stake (PoS) [13] as its consensus mechanism for the mining process. In PoS, a validator, who is responsible for proposing a new block, is randomly selected based on the amount of staked ETH in users' deposits. When the mining process is completed, the valid block is added to the main chain of blocks. After that, the block is irreversible to ensure the integrity of transactions in a blockchain. Another characteristic

of blockchain is transparency which enables all MNs to access the history of transactions within a blockchain network. This transparency ensures total transaction records are visible to all MNs and promotes trust in the blockchain network. Overall, blockchain possesses numerous valuable characteristics, including decentralization, transparency, immutability, and data tamper resistance, making it applicable across various sectors to enhance human life.

### *B. 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 Section V). This network uses the latest version of the Ethereum network (i.e., Ethereum 2.0). This version uses PoS as a consensus mechanism for validating new blocks. Our system includes various MNs, to collect data from their local networks, and bootnodes, the management nodes to connect MNs together. The MNs 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 MNs. They are then put into a block, and the MNs 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, re-entrancy, delegatecall, and function default visibility) happened and caused serious damage to blockchain systems [14]. 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 paper, we consider a blockchain system with $T$ MNs working in a blockchain system as described in Fig. 1. When an MN receives transactions from the blockchain network, it uses **BCEC** to preprocess them by extracting information from important features and then converting them to grey images. After that, we propose a collaborative learning framework for analyzing the images to detect insight attacks in transactions and SCs. In our framework, each MN uses its local dataset to train a deep neural network. After the training process, each MN shares its trained model with other nodes and also receives their trained models in return. Afterward, every MN 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, MN can exchange its learning knowledge with the neural network of other MNs. This approach can not only improve the overall learning knowledge of the neural network of all MNs but also protect the privacy of local data over network transmission. By preventing the transmission of the local data of each MN over the network, our approach can also reduce network traffic to avoid network congestion. Thus, the neural networks of MNs can improve the accuracy of detecting attacks for transactions and SCs in blockchain systems.

## IV. Proposed Attack Detection framework

In our proposed attack detection framework, the MNs are used to learn and share their learning knowledge with others to improve the accuracy of their attack detection. At each MN, we propose to use a deep neural network as a detector to learn the data of the MN's local system. After that, the MN exchanges its learning knowledge (i.e., trained model) with other MNs. When an MN receives trained models from others, it will integrate these models 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 MN can exchange the trained model with others to improve the accuracy of attack detection.

### *A. Preprocessing Process*

Fig. 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 MN for a mining process. From the MN 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 MN and then recover transactions from transaction hashes to have the full information of all transaction features 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 in some specific types of attacks, such as Flooding of Transactions [15], 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 (its benefits are discussed more details in Section V). After that, we apply appropriate preprocessing methods to the corresponding features as follows:
  - **Bytecode feature**: Extract the content and then transform them into opcode using EVM Bytecode Decompiler [16]. The opcode is a series of executed

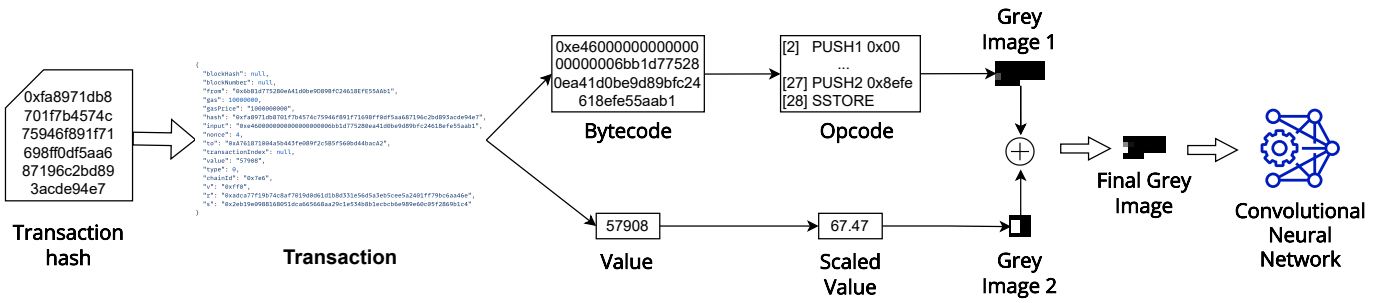



Fig. 2: The preprocessing process of our proposed framework. Our developed BCEC tool first collects the transactions in mining nodes. It then extracts the content of transactions to find "Bytecode" and "Value". After that, this tool converts them into images for further processing.

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.

### *B. Learning Process*

In our proposed framework, at each MN, 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 [17]. Additionally, in our proposed approach, the CNN model does not have to learn their local data separately, it can exchange its trained model with other MNs to improve the learning knowledge as well as enhance the accuracy of attack detection. In detail, the architecture of CNN in an MN includes three types of layers, i.e., convolution layer, max pooling layer, and fully connected layer [17]. Fig. 3 describes the layer of a CNN in an MN. 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 [18]. 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 [17] 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 $\boldsymbol{D}$ as a local dataset of an MN 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 $N$ as the number of training layers of the neural network. We denote $\boldsymbol{I}$ as 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 follows [19]:

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

where $(*)$ is the convolutional operation, $\gamma_n$ 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:

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

where $\varphi$ is the max pooling function that selects the maximum value in a pooling area. We denote $\boldsymbol{I}_{e,i}$ as 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:

$$\begin{aligned} p(\hat{Y}_l = l|\boldsymbol{I}_{e,i}, \boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i}) &= softmax(\boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i}) \\ &= \frac{\exp\left(\boldsymbol{W}_{e,i}\boldsymbol{I}_{e,i} + \boldsymbol{b}_{e,i}\right)}{\sum_l \exp\left(\boldsymbol{W}_{e,l,i}\boldsymbol{I}_{e,i} + \boldsymbol{b}_{e,l,i}\right)}, \end{aligned} \tag{3}$$

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

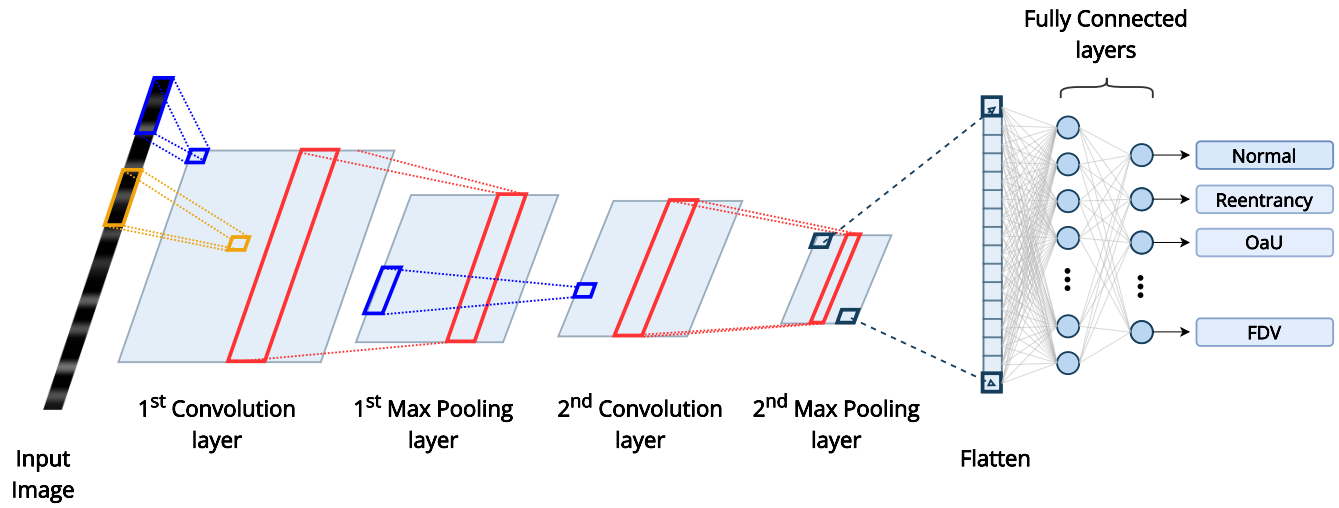



Fig. 3: The architecture of a CNN model. The convolution layer learns the feature representation of the input. The Max pooling layer reduces the resolution of the feature map in the previous layer. The Fully connected layer performs classification functions to produce output.

equation (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:

$$\hat{\boldsymbol{Y}} = \underset{l}{\operatorname{argmax}}[p(\hat{Y}_l = l|\boldsymbol{I}_{e,i}, \boldsymbol{W}_{e,i}, \boldsymbol{b}_{e,i})]. \tag{4}$$

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 $Y_l \in \boldsymbol{Y}$ as the label of class $l$ in $\boldsymbol{Y}$. The loss function can be calculated as follows:

$$\boldsymbol{J}(\boldsymbol{W}) = -\sum_{l=1}^{L} Y_l \log \hat{Y}_l. \tag{5}$$

We denote $\boldsymbol{W}$ as the model of the neural network. Based on equation (5), we can calculate the gradient of this function as follows:

$$\nabla\boldsymbol{\theta} = \frac{\partial \boldsymbol{J}(\boldsymbol{W})}{\partial \boldsymbol{W}} = -\frac{\partial\left(\sum_{l=1}^{L} Y_l \log \hat{Y}_l\right)}{\partial \boldsymbol{W}}. \tag{6}$$

After having the gradient based on equation (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 [20] as $m_{i+1} = A_1(\nabla\boldsymbol{\theta})$ and $v_{i+1} = A_2(\nabla\boldsymbol{\theta})$. We denote $\boldsymbol{\Gamma}_i$ as a trained model, and $\theta_i$ as a global model at iteration $i$. With $\beta_{i+1}$ as the learning rate, a new trained model at the next iteration $i+1$ can be calculated as follows:

$$\begin{aligned} \boldsymbol{\Gamma}_{i+1} &= \boldsymbol{\Gamma}_i - \beta_{i+1}\frac{m_{i+1}}{\sqrt{v_{i+1}}} \\ &= \boldsymbol{\Gamma}_i - \beta_{i+1}\frac{A_1(\nabla\boldsymbol{\theta}_i)}{\sqrt{A_2(\nabla\boldsymbol{\theta}_i)}}. \end{aligned} \tag{7}$$

### C. Collaborative Learning Process

In this paper, 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 MN has a CNN model to train and test its dataset. The CNN model can receive trained models from other MNs to improve the accuracy of attack detection. To do this, the CNN model of an MN first gets the trained model (gradient) based on equation (6). It then sends the trained model to other MNs and receives trained models from others. We denote $T$ as the total number of MNs and $t \in T$ as the MN number. We consider at iteration $i$, an MN receives $T-1$ trained models from others. $\boldsymbol{\theta}_{t,i}$ is the trained model of MN $t$ at iteration $i$. It can aggregate all trained models using the following formula [21]:

$$\boldsymbol{\theta}_{i+1} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\theta}_{t,i}, \tag{8}$$

where $\boldsymbol{\theta}_{i+1}$ is the new aggregated trained model. After generating a new aggregated trained model, each MN will calculate a new trained model using equation (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 MN to analyze and detect the attacks inside a series of grey images. This process is summarized in Algorithm 1.

**Algorithm 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 MN-$t$ learns $D_t$ to produce $\hat{Y}$.
4:  The MN-$t$ creates gradient $\theta_t$ and sends it to others
5:  The MN-$t$ receives $T-1$ gradients from others.
6:  MN calculates a new optimal trained model $\boldsymbol{\Gamma}_{i+1}$.
7:  **end for**
8:  $i = i + 1$.
9: **end while**
10: MN uses its optimal model $\boldsymbol{\Gamma}_{optimal}$ to detect attacks based on input grey images.

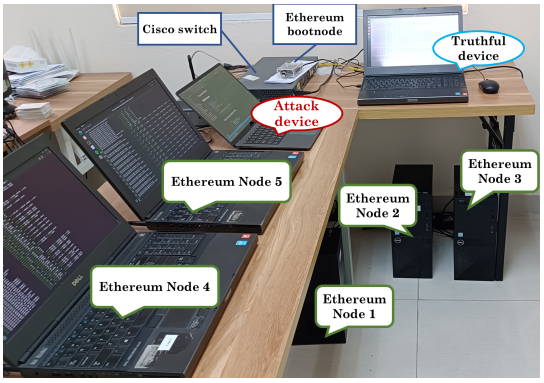



Fig. 4: Real-time experiment setup.

## V. Experiment and Performance Analysis

### A. Experiment Setup

In our experiments, we set up an Ethereum 2.0 system in our laboratory as shown in Fig. 4. This version of Ethereum uses a new consensus mechanism namely 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 [22] and *Prysm v3.2.0* - an official implementation of the PoS consensus mechanism in Ethereum 2.0 [23]. 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 *Prysm v3.2.0*, respectively. They are responsible for connecting all the Ethereum nodes together.

### B. Dataset Collection

According to the detailed analysis of the public Ethereum network on transaction behavior [24], 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 Fig. 4, randomly selects accounts from these accounts to create transactions for the blockchain system.

*1) Normal State:* For the normal state, we use *OpenZeppelin Contracts* [25] 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 SCs' categories 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.

*2) Attack States:* SCs have a number of vulnerabilities listed in SWC [3] because of programmers, consensus mechanisms, and compilers. Attackers can exploit these weaknesses of SC to perform attacks and then steal money in blockchain systems [14]. In this work, we regenerate several real-world attacks from the tracks that they left on Ethereum's ledger. 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 SC GovernMental's 1,100 ETH jackpot payout was stuck [3]. 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, USD $800k of PoWH token [5], and so on [3]. 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 [15]. 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* 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

TABLE I: Number of samples on the proposed BTAD 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 |

dataset [3].

- *Delegatecall (DeC)*: *delegatecall()* is the mechanism to inherit functions, storage, and variables from other deployed SCs. If the inherited SCs are attacked, they will in-directly affect the main SC. To implement, we re-create the $2^{nd}$ Parity MultiSig Wallet attack [3]. 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 [3]. In this attack, attackers took control of this SC through an FDV flaw.

Table I 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*.

### C. Evaluation Methods

The confusion matrix [26], [27] is widely used to evaluate the performance of machine learning models. We denote TP, TN, FP, and TN as "True Positive", "True Negative", "False Positive", and "True Negative". In this paper, 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:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{9}$$

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:

$$\text{Precision} = \sum_{l=1}^{L} \frac{\text{TP}_l}{\text{TP}_l + \text{FP}_l}. \tag{10}$$

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

$$\text{Recall} = \sum_{l=1}^{L} \frac{\text{TP}_l}{\text{TP}_l + \text{FN}_l}. \tag{11}$$

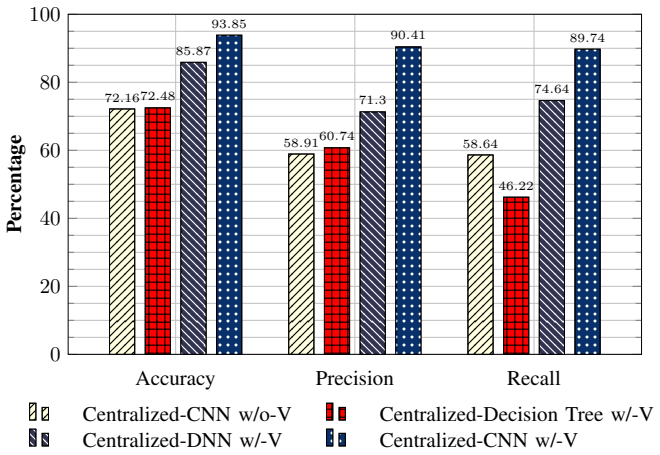



Fig. 5: The results of the preprocessing processes in different schemes.

### D. 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.

*1) Preprocessing Analysis:* In this section, we compare our proposed model in various schemes. In the first hand, we use our proposed preprocessing process as in Fig. 2 in different schemes such as CNN, Deep Neural Network (DNN), Decision Tree (DT) to compare their evaluation results. In the second hand, we eliminate the value feature and use only the Bytecode preprocessing and the CNN 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. Fig. 5 describes the evaluation results of these schemes. In this figure, the model w/-V has accuracy, precision, and recall 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. The DNN and DT schemes with Value feature achieve accuracies of 85.87% and 72.48%, respectively. They are higher than that of The CNN without Value feature at 72.16%, but lower than that of CNN with Value feature at 93.85%. Especially, Fig. 6 provides detailed information for all types of attacks and normal behavior. In Fig. 6, 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 many types of important attacks.

*2) 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 MNs for training and testing processes. Besides, we use different schemes of the collaborative learning model with

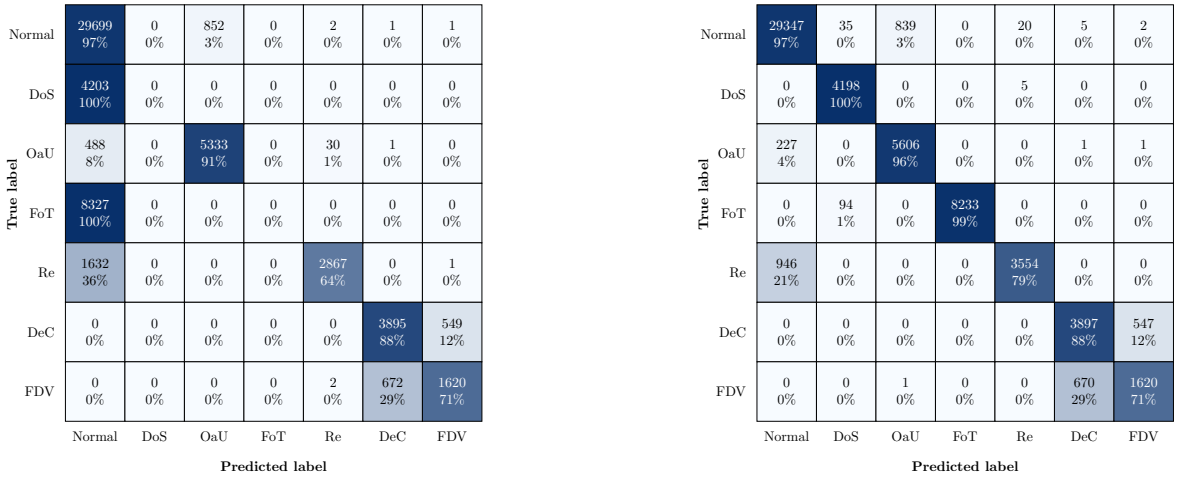



Fig. 6: The detection results of the models w/ and w/o-V feature. (a) Centralized-CNN w/o-V. (b) Centralized-CNN w/-V.

3 mining nodes (Co-CNN-3), 5 mining nodes (Co-CNN-5), and 10 mining nodes (Co-CNN-10). In each scheme, the collected datasets are divided equally among all mining nodes. To implement experiments, we first perform cyberattacks on transactions and SCs in our deployed private Ethereum platform to collect datasets from all MNs. In our proposed collaborative learning model, each MN uses its local dataset for both training and testing processes. However, in the training process, the MNs 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 MNs 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 II and Table III. Table II presents the performance of the simulation results of all schemes with the w/o-V preprocessing process. In Table II, the accuracy, precision, and recall are nearly the same at around 72-73%, 58-59%, and 58-59%, respectively. In contrast, in Table III, 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 III that the performance results of our proposed models are nearly the same as the Centralized-CNN. However, in some MNs such as MN-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, Fig. 7 provides detailed information for each type of attack of the Centralized-CNN and MN-5 of Co-CNN-5. These figures show that the misdetection of MN-5 of the Co-CNN-5 is dramatically reduced compared to the Centralized-CNN. In detail, the misdetection of the MN-5 from Normal to DoS is at 0.88% which is smaller than that of the Centralized-CNN at 1.14%. Similarly, the misdetection of the MN-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%.

*3) 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 mining nodes. Fig. 8 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 MN 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) 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 MNs. 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 IV 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 IV(a) that the performance of these models in accuracy, precision, and recall w/-V in the

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

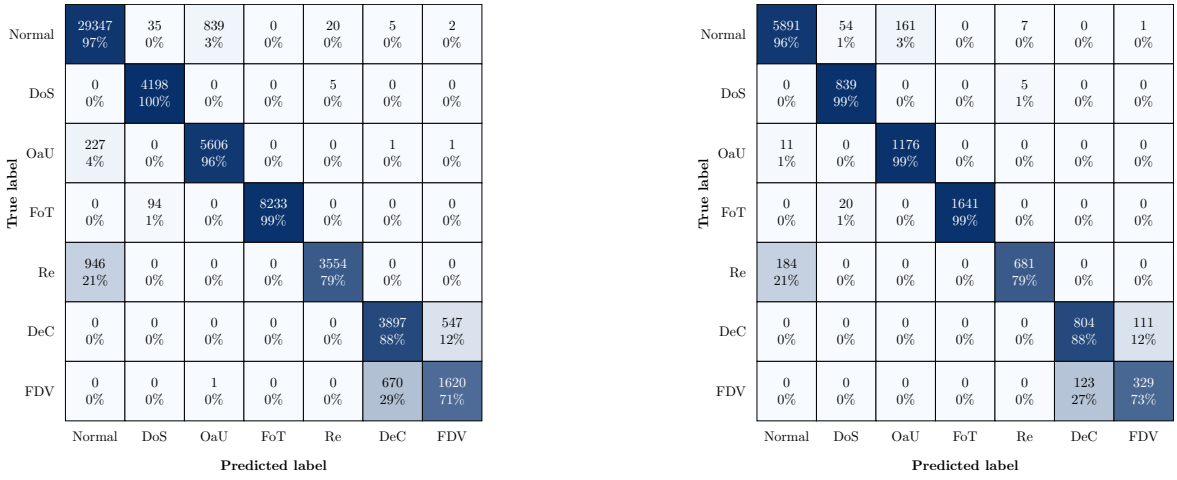
| | Centralized-CNN | Co-CNN-3 | | | Co-CNN-5 | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | MN-1 | MN-2 | MN-3 | MN-1 | MN-2 | MN-3 | MN-4 | MN-5 |
| **Accuracy** | 72.163 | 71.686 | 71.761 | 72.080 | 72.735 | 72.519 | 72.211 | 72.760 | 72.627 |
| **Precision** | 58.911 | 58.323 | 58.298 | 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 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | MN-1 | MN-2 | MN-3 | MN-4 | MN-5 | MN-6 | MN-7 | MN-8 | MN-9 | MN-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 III: 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 | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | MN-1 | MN-2 | MN-3 | MN-1 | MN-2 | MN-3 | MN-4 | MN-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 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | MN-1 | MN-2 | MN-3 | MN-4 | MN-5 | MN-6 | MN-7 | MN-8 | MN-9 | MN-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 |



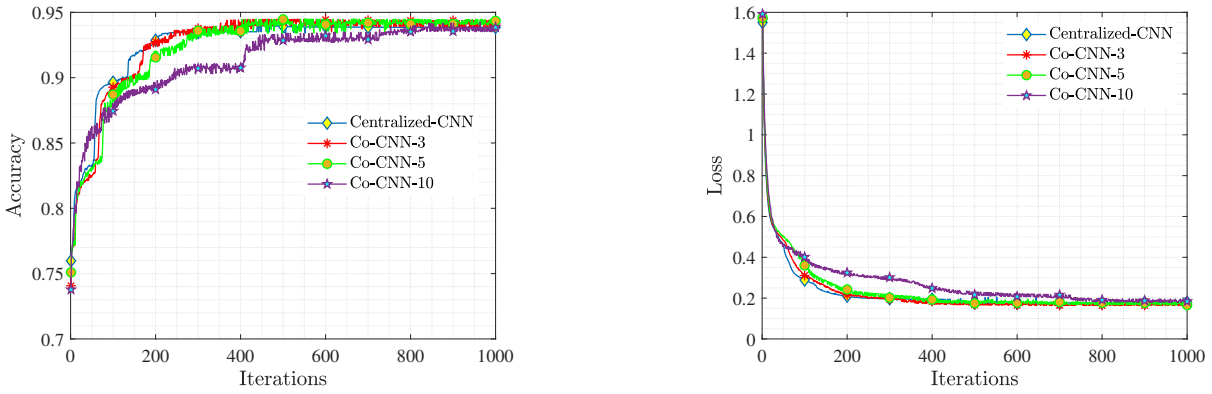

(a) (b)

Fig. 7: The detection results of Centralized-CNN and Co-CNN-5 models. (a) Centralized-CNN w/-V. (b) Co-CNN-5 w/-V.

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 III and the real-time experimental results as in Table IV(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, some 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 IV(a) that MN-4 of Co-CNN-5 has higher performance in accuracy, precision, and recall than MN-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.

*5) Real-time Monitoring and Detection:* Fig. 9 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. Fig. 9(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 Fig. 9(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




(a) (b)

Fig. 8: The convergence of accuracy and loss over iterations: (a) The accuracy over interactions, and (b) The loss over iterations.

TABLE IV: Real-time experiment results.

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

| | Centralized-CNN | | | | | Co-CNN-3 | | | | | Co-CNN-5 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **MN-1** | **MN-2** | **MN-3** | **MN-4** | **MN-5** | **MN-1** | **MN-2** | **MN-3** | **MN-4** | **MN-5** | **MN-1** | **MN-2** | **MN-3** | **MN-4** | **MN-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 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **MN-1** | **MN-2** | **MN-3** | **MN-4** | **MN-5** | **MN-1** | **MN-2** | **MN-3** | **MN-4** | **MN-5** | **MN-1** | **MN-2** | **MN-3** | **MN-4** | **MN-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 |

states in Fig. 9(b), Fig. 9(f) and Fig. 9(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%.

*6) Processing Time:* Fig. 10 describes the processing time of two MNs with the same Co-CNN-5 model. We can observe in Fig. 10 that when the number of transactions increases, the processing time of both MNs also linearly increases. However, there is a different capacity between the two MNs. In detail, while MN-5 can process about 1,100 transactions per second, the number of transactions that MN-1 can process is around 2,150 transactions per second. This is because of the different types of computer configuration between the two MNs described in section V-A. However, in the mainnet of the Ethereum system, the maximum recorded number of transactions is 93.01 per second [28]. Therefore, the capacity of our proposed system can be well-adapted to detect attacks on the mainnet Ethereum system.

## VI. Conclusion

In this work, we developed a collaborative learning model that can efficiently detect malicious attacks in transactions and smart contracts in a blockchain network. To do this, we implemented a private Ethereum network in our laboratory. We then performed attacks in transactions and SCs of that network for analysis. Next, we analyzed the transaction data and extracted the important features (i.e., Bytecode and value) to build the dataset. After that, we converted the dataset into grey images to train and evaluate the performance of our proposed model. In our proposed model, a learning node can detect the attacks in transactions and SCs of a blockchain network and receive and aggregate learning knowledge (i.e., trained models) from other learning nodes to improve the accuracy of detection. In this way, our proposed model does not expose the local data of learning nodes over the network, thereby protecting the privacy of the local data of learning nodes. Both simulation results and real-time experimental results showed the efficiency of our proposed model in detecting attacks. In the future, we will continue studying to develop other methods for detecting attacks in various kinds of networks.

## References

[1] Y. Yuan and F.-Y. Wang, "Blockchain and cryptocurrencies: Model, techniques, and applications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 9, pp. 1421–1428, Sep. 2018.

[2] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," Ethereum Foundation, Tech. Rep., Jan. 2014.

[3] SmartContractSecurity, "SWC Registry - Smart Contract Weakness Classification and Test Cases," Accessed: Sept. 18, 2022. [Online]. Available: https://swcregistry.io

[4] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F.-Y. Wang, "Blockchain-enabled smart contracts: architecture, applications, and future trends," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2266–2277, Feb. 2019.

[5] E. Banisadr, "How $800k Evaporated from the PoWH Coin Ponzi Scheme Overnight," Accessed: Feb. 13, 2023. [Online]. Available: https://medium.com/@ebanisadr/how-800k-evaporated-from-the-powh-coin-ponzi-scheme-overnight-1b025c33b530

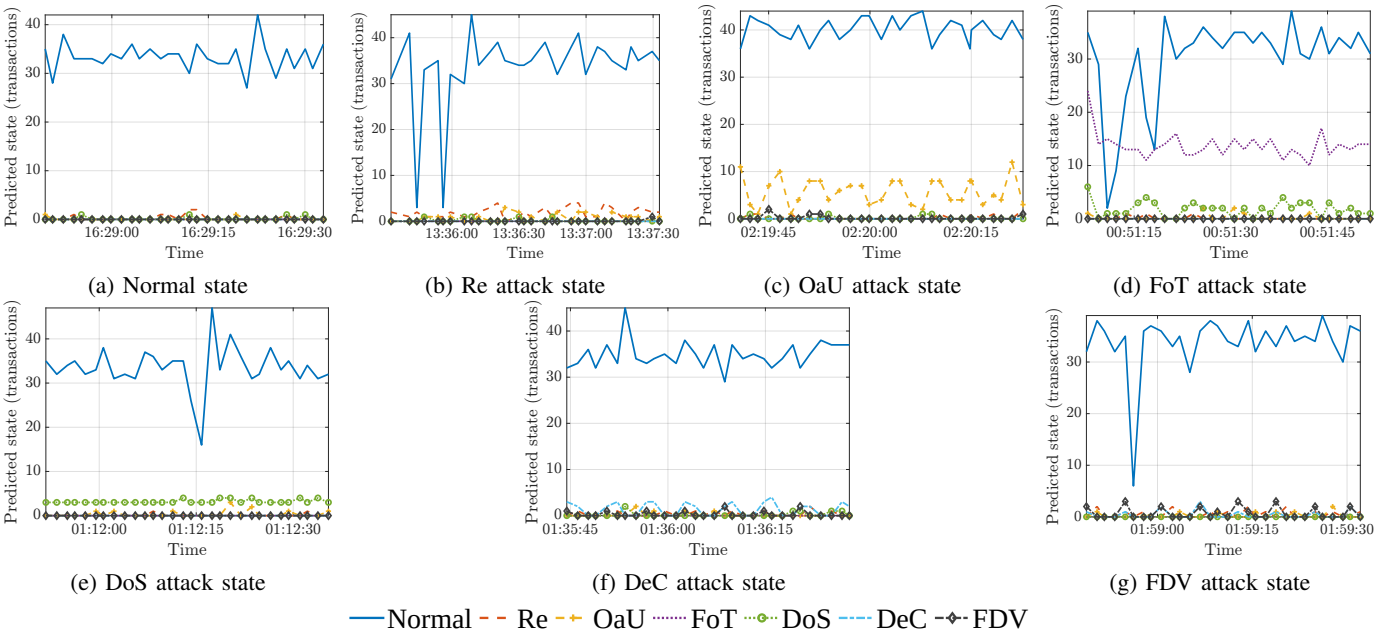



Fig. 9: Real-time cyberattack detection: proposed Co-CNN-5 model in Ethereum node 1.

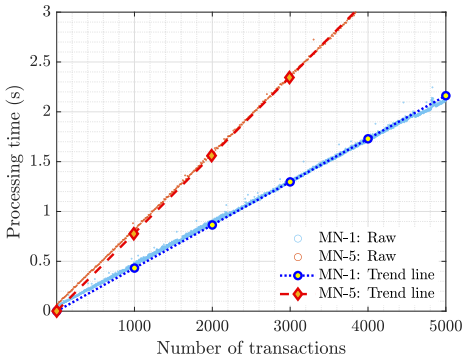



Fig. 10: Throughput of proposed Co-CNN-5 model in two computer configurations.


[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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, Vietnam, Nov. 2019, pp. 485–505.

[11] 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.

[12] B. Lashkari and P. Musilek, "A comprehensive review of blockchain consensus mechanisms," *IEEE Access*, vol. 9, pp. 43 620–43 652, Mar. 2021.

[13] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining ghost and casper," May 2020. [Online]. Available: https://arxiv.org/abs/2003.03052

[14] 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.

[15] 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*, Feb. 2024, (accepted).

[16] L. Hollander, "Evm bytecode decompiler," Accessed: Feb. 10, 2023. [Online]. Available: https://www.npmjs.com/package/evm

[17] 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.

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

[19] 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.

[20] 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.

[21] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," Oct. 2016. [Online]. Available: https://arxiv.org/abs/1610.02527

[22] Ethereum, "Official Go implementation of the Ethereum protocol," Accessed: Nov. 18, 2022. [Online]. Available: https://github.com/ethereum/go-ethereum/tree/v1.10.22

[23] Prysmatic Labs, "Prysm: An Ethereum Consensus Implementation Written in Go," Accessed: Jan. 10, 2023. [Online]. Available: https://github.com/prysmaticlabs/prysm/tree/v3.2.0

[24] 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.

[25] OpenZeppelin, "A library for secure smart contract development," Accessed: Sept. 18, 2022. [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts

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

[27] 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.

[28] Etherscan, "Ethereum daily transactions chart," Accessed: Feb. 10, 2023. [Online]. Available: https://etherscan.io/chart/tx