Developing Customized & Secure Blockchains with Deep Federation Learning to Prevent Successive Attacks

Soumya Banerjee, Soham Chakraborty, Paul Mühlethaler

▶ **To cite this version:**

Soumya Banerjee, Soham Chakraborty, Paul Mühlethaler. Developing Customized & Secure Blockchains with Deep Federation Learning to Prevent Successive Attacks. MSPN 2020 - 6th International Conference on Mobile, Secure and Programmable Networking, Oct 2020, Paris / Virtual, France. hal-02997482

HAL Id: hal-02997482
https://hal.science/hal-02997482

Submitted on 10 Nov 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Developing Customized & Secure Blockchains with Deep Federation Learning to Prevent Successive Attacks

[1]Soumya Banerjee[1], [2]Soham Chakraborty, [3]Paul Mühlethaler
[1]Trasna-Solutions, Ireland, [2]KIIT, Bhubaneshwar, India, ,[3]EVA Project, Inria - Paris, 75012 Paris, France

Abstract

Recently, blockchain technology has been one of the most promising fields of research aiming to enhance the security and privacy of systems. It follows a distributed mechanism to make the storage system fault-tolerant. However, even after adopting all the security measures, there are some risks for cyberattacks in the blockchain. From a statistical point of view, attacks can be compared to anomalous transactions compared to normal transactions. In this paper, these anomalous transactions can be detected using machine learning algorithms, thus making the framework much more secure. Several machine learning algorithms can detect anomalous observations. Due to the typical nature of the transactions dataset (time-series), we choose to apply a sequence to the sequence model. In this paper, we present our approach, where we use federated learning embedded with an LSTM-based autoencoder to detect anomalous transactions.

Keywords – Blockchain; Anomaly detection; Intelligent Algorithm; Autoencoders; Sequence-to-sequence models; Federated Learning;

1. Introduction

Blockchain is a peer-to-peer, distributed ledger technology (DLT), which stores data/transactions in a distributed data-structure. Anyone on the network can explore and participate in the transactions on the blockchain in a secure manner. Several techniques are involved, such as a consensus mechanism and block mining . Although certain techniques make it more secure and fault tolerant (e.g. Byzantine fault tolerance), certain malicious and untrusted nodes/users still can persist. They can severely affect the transactions and interrupt the security of the system [1]. The two main challenging security issues are double spending [2] and record hacking (or record manipulating). There are several non-intelligent algorithms for checking whether a transaction involves double spending or a transaction is being hacked (e.g. maintaining transaction id, hashing and cryptographic encryption). However, they seldom fail to investigate

[1] Associated Senior Researcher, INRIA-EVA, Paris

the nature of malicious nodes and may lead to blocks getting hacked/affected. Therefore, we can switch to using machine learning driven intelligent algorithms [3] to detect the possibility of whether a particular node is malicious or not . Predicting this phenomenon will clearly be advantageous, as any malicious transactions can be halted at that instant without the possibility of a spoofing attack. The remaining part of this paper is as follows:
Section 2 briefly describes recent research breakthroughs in the field of machine learning (ML) algorithms. The proposed model is presented in Section 3, with while Section 3.1 giving the high level description of the algorithm with the relevant mathematical parameters. Section 4 discusses the experimental results yielded through the proposed model and finally section 5 summarizes the contribution as conclusion with future possibilities of extension of research in this regard.

2. Related Work

This paper is motivated by the ideas and implementations of reinforcement learning towards enhancing the security of blockchains [4] [5]. However, most studies tend to offer consolidated reviews of blockchain, irrespective of blockchain implementations. Thus, the authors in [5] point out that the principal of optimization in the context of blockchains enabled the Internet of Vehicles (IoV). In contrast, this present work sets out to achieve two main goals : a) to represent a distributed training algorithm that applies an autoencoder to detect anomalies in the chain of transactions in different nodes. The autoencoder uses LSTM for sequence-to-sequence learning and the Adam optimization algorithm to optimize the parameters. This choice was made since Adam optimizers usually work better than RMSprop and SGD.
b) to analyze the complexity of blockchain processes
In parallel, the proposed model also executes the ML algorithm to investigate suspicious spikes that appear during the process of creating the blocks (it is known as the Growth model). Let α and β be positive and non-negative probability distributions.
Let t_k represent the (absolute) time at which block k is created, h_k the length of the local blockchain after being extended with block k, and z_k the cumulative maximum given by z_k: =max $\{h_i \mid i \leq k\}$. The overall complexity of conventional O(mn) is due to the computation of any functional matrix d and its time complexity is O $(nm + n^2)$. It should be noted that there are n iterations for any blockchain process, each requiring O(n) and O(m) time to compute h_k and d_k, respectively. However, if a single fast algorithm is used to compute h_k, the average overall complexity

is reduced. In the worst-case scenario, the complexity is O(k). Here, the experimental evaluations suggest an average below O (β/α) (constant with respect to k). Thus, the average runtime complexity is bounded by [O nm + min {n^2, n + nβ/α}], and this corresponds to O(nm), unless the blockchain system is extremely fast (β≫α).

This paper subjects a challenging proposition over this typical complexity evaluation of blockchain processes. In fact, the proposed model considers the total process, including the complexity of ML with this present blockchain complexity in parallel mode. Therefore, the objective is to synchronize the process of intelligent algorithm in parallel mode with the master blockchain algorithm.

3 Proposed Model

As a primary step, a minimum number of transactions are collected as data to train the designated neural model on these transactions. We can arrange the transactions in time window frames - a set of temporal transaction vectors will be considered as a single observation point. As the data mostly contains a balanced mixture of proper and non-malicious transactions without relevant labels, we need to deploy an unsupervised learning technique. The malicious transactions can be found by using the encoder-decoder model. Here, the n-dimensional transactions will be cast into a latent space. As the malicious activities will comprise different patterns/trends than normal transactions(it is assumed that the number of normal transactions is probably greater in number), therefore they will be considered as outliers in the latent space. Once the decoder has been applied we will again retrieve the n-dimensional search space containing the non-malicious transactions. By comparing the previous search space with the generated one, we can identify the malicious transactions. Once the basic autoencoder model is trained, it is put into the distributed setting. Here, we incorporate deep federated learning [9] for the purpose of real-time distributed learning. The transaction data generated at every connected client/node participating in the blockchain will be used to train the federated model. Finally, the weights of the client models will be updated and aggregated in the master model. Initially the pre-trained encoder decoder model is set as the master model in the federated learning setup. At every remotely connected nodes, a client model will be responsible and prepared, which will use the weights of the master model at the beginning. With time, transactions occur at nodes and the corresponding client node updates it weights. After a particular interval the aggregate of all the client node model's weights is sent to the master model for a final update.

There are certain basic features that have a correlation with the type of

transactions – whether they are malicious or not. These are :
• time of transaction,
• frequency of transaction,
• sending transaction id,
• receiving transaction id.

A holistic view of functional flow is presented in Figure 1.

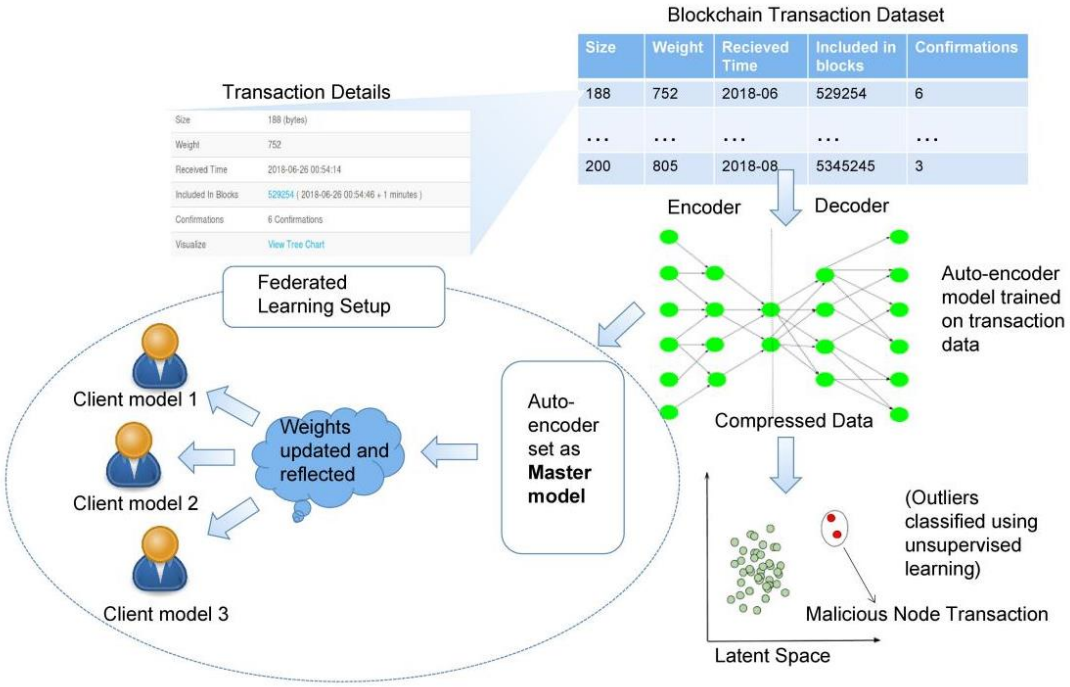


Figure 1: High-level functional flow

3.1 Algorithm

Input Data: The training dataset contains N_F features and N_O number of observations. We take N_W number of temporally consecutive observations as a single observation window. So the input matrix $X \in R^{N_O * N_F * N_W}$

Output: Trained encoder parameters (θ) and decoder parameters (ϕ)

Algorithm : Training Algorithm(X, θ, ϕ)

1: Initialize number of training loops as N_T
2: for i $\in$ { 1, 2, …, N_T } do
3: Select a window from training set; $X^W \subset X : X^W \leftarrow [X_S, X_{S+1}, \ldots, X_{S+Nw}]$
4: s $\leftarrow$ s+1

5: Initialize encoder LSTM loop : $h_{j\theta}^{2} \leftarrow f_{LSTM\,\theta}(h_{j\theta}^{1})$ for $j \in \{1,2,3,\ldots N_W\}$

6: for n $\in$ { 2, 3, …, N_W } do

7: $h_{j\theta}^{n+1} \leftarrow f_{LSTM\,\theta}(h_{j\theta}^{n})$; for $j \in \{1,2,3,\ldots N_W\}$

8: Initialize decoder LSTM loop : $h_{j\phi}^{2} \leftarrow f_{LSTM\,\phi}(h_{j\phi}^{1})$ for $j \in \{1,2,3,\ldots N_W\}$

9: for n $\in$ { 2, 3, …, N_W } do

10: $h_{j\phi}^{n+1} \leftarrow f_{LSTM\,\phi}(h_{j\phi}^{n})$; for $j \in \{1,2,3,\ldots N_W\}$

11: Objective Function : $J(\theta,\phi) \leftarrow -\sum_{w=1}^{|N_W|}\sum_{f=1}^{|N_F|} \| X_{w,f}^{W} - X'^{W}_{w,f} \|_2$

12: Calculate gradient : $G \leftarrow \frac{dJ(\theta,\phi)}{d(\theta,\phi)}$

13: Update parameters : $\theta,\phi \leftarrow$ ADAM(G)

14: end for

Notation	Interpretation
N_T	Number of training loops
X^W	Window frame vector
N_W	Number of transactions in a window frame
$h_j\theta$	hidden parameters for encoder
$h_j\phi$	hidden parameters for decoder
N_F	Number of features
J	Objective function
G	Gradient

Table 1: Descriptions of Notations used in training algorithm

Algorithm: Federated Learning algorithm

Local window frame size N_W, number of participants m per iteration, number of local epochs E.

Randomly initialize the parameters θ_G, ϕ_G

1: [Participant i]
2: LocalTraining(i, θ, ϕ):
3: Split local dataset Di to consecutive temporal window frame of size N_W
4: Training Algorithm(Di, θ, ϕ)
5:
6: [Master Node]
7: Initialize $\theta^0{}_G, \phi^0{}_G$
8: for each iteration t from 1 to T do
9: Randomly choose a subset St of m participants from N
10: for each partipant i ∈ St parallely do
11: $\theta^{t+1}{}_i, \phi^{t+1}{}_i \leftarrow$ LocalTraining(i, $\theta^t{}_G, \phi^t{}_G$)
12: end for
13: $\theta^t{}_G, \phi^t{}_G = \frac{1}{\sum_{i \in \mathcal{N}} D_i} \sum_{i=1}^{N} D_i * (\theta_i^t, \phi_i^t)$
14: end for

Notation	Interpretation
M	number of participants considered for weight aggregation
θ_G, ϕ_G	Global parameters
E	Number of local epoch
θ, ϕ	Local parameters

Table 2: Notations used in the Federated Learning algorithm

In the original dataset, every individual transaction event is counted in several window frames. So the degree of outliers in each of them can be calculated by calculating the mean error between the predicted output vector and the input vector that contains the particular transaction event.

$$score(\overrightarrow{X_t}) = \frac{1}{N_W}\sum_{i=1}^{N_W} || X^W - X'^W ||_2 \quad (1)$$

where i represents the index of all the window frames that contain transaction X_t, where X^W is the actual frame and X'^W is the predicted frame.

4 Results and Discussion

In this proposed methodology, a distributed autoencoder model is set that can summarize the state of the ledger, on a latent space and can itself recreate the actual information from the latent space. The underlying idea of this methodology is that whenever the state of the transactions is consistent, the autoencoder preserves the original information from the space. On the other hand, anomalous situations contain inconsistent properties and values that result in unsuccessful reconstruction of the original information. Let's consider an instance where the amount of transactions is too high compared to all other attributes of the transaction. The autoencoder will represent this value as a noise and will automatically ignore this at the time of reconstruction. In such cases, the differences between the actual values and the recreated values depict the score of outliers, which in turn depicts the degree of anomalous issues over the transaction. Therefore, the transactions that have abnormally high outlier scores will be considered as suspicious transactions.

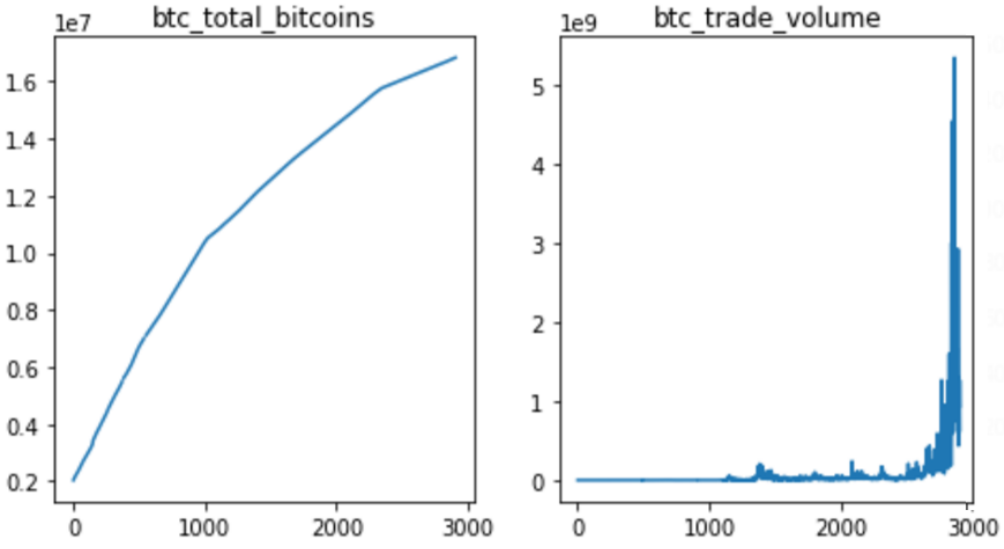
The dataset was collected from github [10]. This dataset was created on historical transactions of BitCoin. The dataset contains 2906 samples with 24 attributes. The attributes of the dataset are listed in the table below (Table 3).

Features
Date
btc_market_price
btc_total_bitcoins
btc_market_cap
btc_trade_volume
btc_blocks_size

btc_avg_block_size
btc_n_orphaned_blocks
btc_n_transactions_per_block
btc_median_confirmation_time
btc_hash_rate
btc_difficulty
btc_miners_revenue
btc_transaction_fees
btc_cost_per_transaction_percent
btc_cost_per_transaction
btc_n_unique_addresses
btc_n_transactions
btc_n_transactions_total
btc_n_transactions_excluding_popular
btc_n_transactions_excluding_chains_longer_than_100
btc_output_volume
btc_estimated_transaction_volume
btc_estimated_transaction_volume_usd

Table 3: Features of the dataset

Some of the previously studied trends/patterns of certain attributes are shown in Figure 2. In the dataset, some of the features had missing values. To get an idea of the trend for interpolation, those features are plotted, the missing data being replaced by forward filling method.



(a) (b)

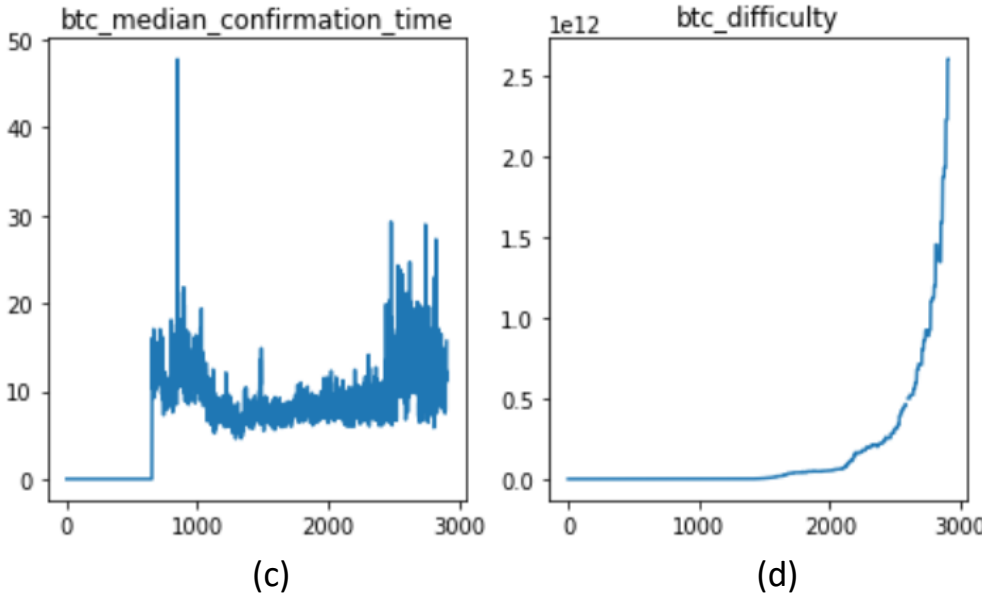


Figure 2 : Exploratory Data Analysis

As discussed in the previous section, we have divided the data into several time frames consisting of time-based transaction events. These individual time frames were trained in the model, and an illustration of the training loss curve is given represented in Figure 3.

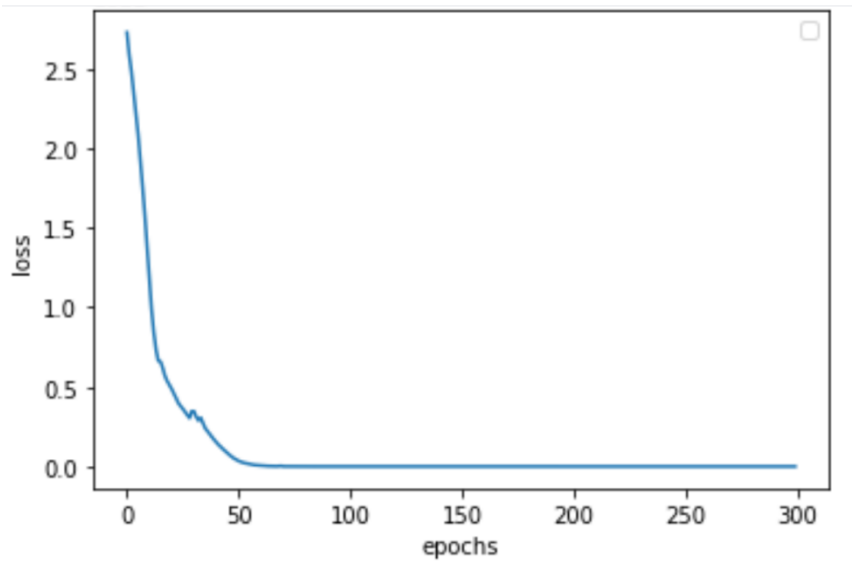


Figure 3 : Loss Curve on training data

Finally, after the model has been trained we try to analyse the score of the outliers. Here we analysed on the first 91 transaction events. Every temporal window considered consisted of three transactions. Once the autoencoder generated a new set of values for every window frame, it was compared with the original timeframe values and the score was calculated using Equation 1. The scores are graphically displayed in

Figure 4. It can be clearly understood from the pictorial view that some of the transactions have an abnormally high outlier score. These transactions can be considered as suspicious transactions.

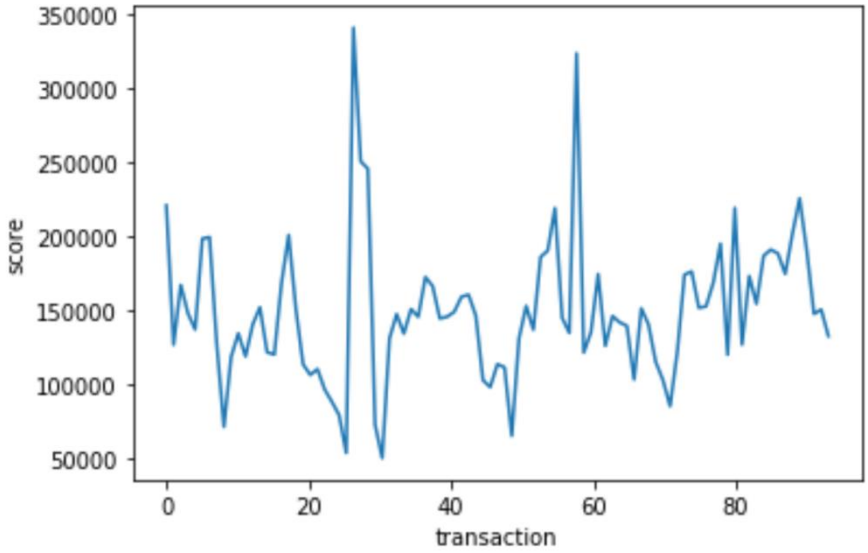


Figure 4 : Score for outliers in transaction events

5. Conclusion & Scope of Future Research

The main objective of this work was to detect an attack in a blockchain network using federated learning embedded with a sequential autoencoder model. As attacks are very rare among many transactions, it is very difficult for anyone to label them manually. We considered an unsupervised learning mechanism in a distributed framework. The proposed model can be used to detect successive attacks beforehand with the master-client mechanism of the federated learning system. In a distributed manner, the model is trained on several client machines and after every interval, the weights of the master model are updated. As soon as any transaction falls as an outlier, it is predicted to be an attack or a suspicious transaction. Using LSTM instead of generic RNN for our training algorithm, we reduce the possibility of vanishing and exploding gradient as the amount of data is large. The sequence-to-sequence deep learning model helps to capture the underlying probability distribution for normal consistent transactions.

The work has manifold future possibilities to integrate ML algorithms in blockchain processes [6] [7] [8] [11]. It is worth investigating deep learning deployment for energy perspectives blockchain. Therefore, the parallel mode and optimized approach of block mining time with the

detection of suspicious blocks might lead to sound synchronization of blockchain process and such distributed machine learning interfaces.

References

1. C. Ye, G. Li, H. Cai, Y. Gu, and A. Fukuda. Analysis of security in blockchain: Case study in 51%-attack detecting. In 2018 5th International Conference on Dependable Systems and Their Applications (DSA), pages 15–24, 2018.
2. W. Chohan. The double spending problem and cryptocurrencies.SSRN Electronic Journal, 01 2017.
3. A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys Tutorials, 18(2):1153–1176, 2016.
4. Reinforcement Learning in Blockchain- Enabled IIoT Networks: A Survey of Recent Advances and Open Challenges, MDPI Sustainability : 24 June 2020.
5. Liu, M.; Teng, Y.; Yu, F.R.; Leung, V.C.; Song, M. Deep Reinforcement Learning Based Performance Optimization in Blockchain-Enabled Internet of Vehicle. In Proceedings of the ICC 2019. 2019 IEEE International Conference on Communications (ICC), Shanghai, China.
6. Thang N Dinh and My T Thai. "AI and Blockchain: A Disruptive Integration. eng. In: Computer 51.9 (2018), pp. 48–53. ISSN: 0018- 9162
7. A. Juneja and M. Marefat. Leveraging blockchain for retraining deep learning architecture in patient-specific arrhythmia classification. In: 2018 IEEE EMBS International Conference on Biomedical Health Informatics (BHI). Mar. 2018, pp. 393–397. doi: 10.1109/BHI.2018. 8333451
8. M. Shen et al. Privacy-Preserving Support Vector Machine Training Over Blockchain-Based Encrypted IoT Data in Smart Cities. In: IEEE Internet of Things Journal 6.5 (Oct. 2019), pp. 7702–7712. ISSN: 2372-2541.
9. Yuzheng Li, Chuan Chen, Nan Liu, Huawei Huang, Zibin Zheng, Qiang Yan. A Blockchain-based Decentralized Federated Learning Framework with Committee Consensus, 2020.
10. https://github.com/Yrzxiong/Bitcoin-Dataset
11. M. A. Ferrag and L. Maglaras. DeepCoin: A Novel Deep Learning and Blockchain-Based Energy Exchange Framework for Smart Grids. In: IEEE Transactions on Engineering Management (2019), pp. 1–13. ISSN: 1558-0040. doi: 10.1109/TEM.2019.2922936.

Appendix

source code:

```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# import the dataset
data = pd.read_csv('bitcoin_dataset.csv')
test = pd.read_csv('test_set.csv')

# plotting some of the features with missing value
%matplotlib inline
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

axes[0].plot(data['btc_total_bitcoins'])
axes[0].set_title("btc_total_bitcoins")

axes[1].plot(data['btc_trade_volume'])
axes[1].set_title("btc_trade_volume")

axes[2].plot(data['btc_blocks_size'])
axes[2].set_title("btc_blocks_size")

fig, axes = plt.subplots(1, 3, figsize=(12, 4))

axes[0].plot(data['btc_median_confirmation_time'])
axes[0].set_title("btc_median_confirmation_time")

axes[1].plot(data['btc_difficulty'])
axes[1].set_title("btc_difficulty")

axes[2].plot(data['btc_transaction_fees'])
```

```
axes[2].set_title("btc_transaction_fees")

# filling the missing data with forward fill method
X = data.fillna(method='ffill')

#creating window frames from the temporally consecutive
transactions

l1=[]
for i in range(2,93):
        l = []
        l.append(list(X.iloc[i-2]))
        l.append(list(X.iloc[i-1]))
        l.append(list(X.iloc[i]))
        l1.append(l)

sequence = np.array(l1)
n_in = len(sequence)
#resahping the flattened array
sequence = sequence.reshape((n_in, 3, 22))

# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(3,22)))
model.add(RepeatVector(3))
model.add(LSTM(100, activation='relu',
return_sequences=True))
model.add(TimeDistributed(Dense(22)))
model.compile(optimizer='adam', loss='mse',
metrics=['accuracy'])

# fit model
history = model.fit(sequence, sequence, epochs=300,
verbose=0)

t = model.predict(sequence)
```

```
# plotting the loss curve of the model
plt.plot(history.history['loss'])
plt.legend()
xlabel('epochs')
ylabel('loss')

# storing the actual transactions
actual=[]
actual.append(list(sequence[0][0]))
actual.append(list(sequence[0][1]))
actual.append(list(sequence[0][2]))

for i in range(1,len(sequence)):
        actual.append(list(sequence[i][2]))

# storing the predicted transactions
pred=[]
pred.append(list(t[0][0]))
pred.append(list(t[0][1]))
pred.append(list(t[0][2]))

for i in range(1,len(t)):
        pred.append(list(t[i][2]))

# finding the error between the actual and the predicted
transactions
error=[]
for i in range(len(pred)):
        e = 0
        for j in range(22):
                e = e + pow(abs(pred[i][j]*pred[i][j] -
actual[i][j]*actual[i][j]),(1/2))
        error.append(e)

# plotting the outlier score vs transaction curve
import matplotlib.pyplot as plt
x = np.linspace(0,93,93)
```

```
plot(x,error)
xlabel('transaction')
ylabel('score')
```