electronics

MDPI

Article

Malicious Contract Detection for Blockchain Network Using Lightweight Deep Learning Implemented through Explainable AI

Yeajun Kang, Wonwoong Kim, Hyunji Kim, Minwoo Lee, Minho Song and Hwajeong Seo *

Division of IT Convergence Engineering, Hansung University, Seoul 02876, Republic of Korea; 22213202@hansung.ac.kr (Y.K.); 22213203@hansung.ac.kr (W.K.); 22111401@hansung.ac.kr (H.K.); 23213701@hansung.ac.kr (M.L.); 23213704@hansung.ac.kr (M.S.)
* Correspondence: hwajeong@hansung.ac.kr; Tel.: +82-760-8033

Abstract: A smart contract is a digital contract on a blockchain. Through smart contracts, transactions between parties are possible without a third party on the blockchain network. However, there are malicious contracts, such as greedy contracts, which can cause enormous damage to users and blockchain networks. Therefore, countermeasures against this problem are required. In this work, we propose a greedy contract detection system based on deep learning. The detection model is trained through the frequency of opcodes in the smart contract. Additionally, we implement Gredeeptector, a lightweight model for deployment on the IoT. We identify important instructions for detection through explainable artificial intelligence (XAI). After that, we train the Greedeeptector through only important instructions. Therefore, Greedeeptector is a computationally and memory-efficient detection model for the IoT. Through our approach, we achieve a high detection accuracy of 92.3%. In addition, the file size of the lightweight model is reduced by 41.5% compared to the base model and there is little loss of accuracy.

Keywords: smart contract; greedy contract detection; deep learning; explainable artificial intelligence; lightweight

Citation: Kang, Y.; Kim, W.; Kim, H.; Lee, M.; Song, M.; Seo, H. Malicious Contract Detection for Blockchain Network Using Lightweight Deep Learning Implemented through Explainable AI. *Electronics* **2023**, *12*, 3893. https://doi.org/10.3390/electronics12183893

Academic Editors: Mikolaj Karpinski, Oleksandr O. Kuznetsov and Roman Oliynykov

Received: 16 August 2023
Revised: 12 September 2023
Accepted: 12 September 2023
Published: 15 September 2023

1. Introduction

Blockchain has recently been used in various fields [1–3]. The reason why blockchain can be used in such a variety of fields is because of smart contracts. A smart contract is a digital contract based on blockchain. Smart contracts allow parties to conclude contracts without the involvement of a third party. However, as a result of analyzing some smart contracts deployed on Ethereum, it was confirmed that greedy contracts are included [4]. A greed contract locks the Ether indefinitely so that it cannot be withdrawn within the contract. Therefore, executing a greedy contract causes enormous damage. There can be events in which the parity multisig wallet is frozen. Due to a code flaw discovered in the multisig wallet library smart contract, approximately 510,000 Ether users were locked indefinitely. Since these problems pose a very critical threat to the blockchain network, they must be prevented.

To prevent this problem, methodologies are needed to prevent malicious activity. There are two main ways to detect malicious activity: malicious node detection and malicious smart contract detection. Among the latter, various malicious smart contract detection models have been proposed [5–7] to detect malicious smart contracts. Most deep-learning-based methods detect malicious smart contracts by learning the features of smart contracts. In addition, research has been conducted to image smart contracts and train them to implement a detection model [8].

However, since the transaction speed is directly related to the scalability of the blockchain, the transaction speed in the blockchain must also be considered. If detection

takes too much time, it incurs computational and memory overhead. However, existing deep-learning-based detection models are designed only to increase accuracy. Therefore, existing deep learning detection models may cause computational and memory overhead. Moreover, a lot of research on IoT blockchain has been conducted recently [9–11]. Therefore, a lightweight detection model as well as a detection rate is essential. In other words, a lightweight deep learning detection model that does not reduce the scalability of blockchain is needed.

In this paper, through XAI, we identify important instructions when deep learning detects greedy contracts. We propose a `Greedeeptector` trained through important instructions. `Greedeeptector` is a lightweight detection model for the Internet of Things (IoT). `Greedeeptector` identifies greedy contracts in a computationally and memory-efficient manner without compromising the scalability of the blockchain or detection accuracy. The efficiency of speed and memory is very important for low-end devices, such as IoT devices. Also, in terms of blockchain scalability, the lower the computational and memory usage, the higher the scalability.

1.1. Contribution

1.1.1. In-Depth Analysis of Greedy Contract Instruction Using Explainable Artificial Intelligence

Through integrated gradient and gradient SHAP, instructions that affect the prediction of the model are identified. We select several important instructions for benign and greedy smart contracts, and we analyze them in depth. In addition, we analyze which instructions have important characteristics and are frequently used in greedy contracts.

1.1.2. Implementation of a Lightweight Neural Network Based on Important Instructions

Important instructions extracted through XAI have fewer data dimensions. We implement a lightweight model using these important instructions for the training process. The lightweight model is about 50% lighter than the previous model. In addition, the lightweight model shows little loss of accuracy.

1.1.3. Improving the Stability of Blockchain Networks through Detection When Executing Contracts

Unlike previous work, our work performs greedy contract detection when the smart contract is executed. That is, detection can be performed not only for newly deployed smart contracts but also for already deployed smart contracts. Thus, it improves the stability of the blockchain network.

The remainder of this paper is organized as follows: In Section 2, related technologies, such as artificial neural networks, smart contracts, and previous work, are presented. In Section 3, the proposed method to detect greedy contracts is introduced. In Section 4, a comparison between the default model and the lightweight model and an in-depth analysis of the contract's instructions are described. Finally, Section 5 concludes the paper.

2. Related Works

2.1. Artificial Neural Networks

An artificial neural network [12] refers to a computer-implemented structure of the neurons in the human brain. Deep learning performs learning by stacking multiple layers of artificial neural networks and consists of an input layer, a hidden layer, and an output layer. The input layer refers to the layer that receives the data to be learned. The hidden layer calculates the weight, and the final result is output through the output layer. Unlike machine learning, deep learning extracts and learns features from data by itself. Due to these characteristics, deep learning is used in various fields, such as computer vision, speech recognition, natural language processing, and signal processing. Multi-layer perceptron (MLP) and convolutional neural networks (CNN) [13] are representative deep learning models and are mainly used for classification problems. In addition, recurrent neural

networks (RNNs) [14], which are good for training time-series data, and generative neural networks, such as generative adversarial networks (GANs) [15], also exist.

2.2. Lightweight Deep Learning

Lightweight deep learning is deep learning that reduces the size of the model and reduces computational complexity while maintaining the performance of models with deep and complex layers [16]. If a non-lightweight deep learning model is used in a low-end IoT device, memory overhead may occur in the process of loading numerous parameters. Therefore, such lightweight deep learning is essential for low-end IoT devices with limited computing resources, such as mobile environments, autonomous vehicles, and robots. We utilize XAI to implement lightweight models.

2.3. Explainable Artificial Intelligence (XAI)

Artificial intelligence is a black-box model and inside the model learns complex relationships and characteristics about data. Therefore, it is not clear what the reasons and grounds for decisions made by AI models are. However, in fields where important decisions must be made, such as medicine, finance, and law, a clear basis for the AI output may be required. XAI [17] is a technology created to solve these problems. XAI is a technology that provides the reasons and rationale for decisions made by deep learning models. XAI can increase the reliability of deep learning technology and has the advantage of facilitating debugging to achieve the result requiired [18]. In addition, by improving understanding of the training result, it can be developed into a better model. XAI can determine which features are needed for prediction. That is, it is possible to learn a deep learning model with necessary features only, excluding unnecessary features. In this way, the deep learning model can be optimized and a lightweight model can be implemented. Therefore, XAI is useful for implementing efficient and lightweight models. We used Captum [19] to apply various algorithms. Captum is a unified, open-source model interpretability library for PyTorch. There are several methods of XAI (e.g., integrated gradients and gradient SHAP). We utilize these two algorithms to implement a lightweight deep learning model.

Integrated gradients (IG) [20] is an XAI algorithm that calculates the importance of each feature by accumulating and multiplying the gradient information and the difference between the input and baseline. Since IG calculates the importance of each feature locally, it does not have a global interpretation function. IG satisfies the sensitivity and implementation invariance conditions. The sensitivity condition states that, if the difference between the baseline and the input is only one feature, the model should have non-zero attribution if it makes different predictions for the two inputs. The implementation invariance condition is a condition that when different models predict the same for the same input, the two models must have a constant attribution for the same input. IG has the advantage of being simple to implement and can be applied to various datasets, such as text and images.

Gradient SHAP [21] is an XAI algorithm that approximates Shapley values through gradients. The Shapley value quantifies the contribution of each player participating in the game based on game theory. In other words, the Shapley value represents the contribution of a specific player when they cooperate with all other players. Recently, it has mainly been used to interpret deep learning models. This allows us to interpret how the features contribute to the output. The Shapley value is useful for feature analysis because it can be analyzed both locally and globally. In addition, the Shapley value has the advantage of high accuracy by considering the interaction between inputs. However, it also has the disadvantage that the computation increases exponentially as the number of features increases. To overcome this shortcoming, Gradient SHAP calculates the Shapley values through gradients.

2.4. Blockchain

Blockchain is a peer-to-peer distributed ledger network in which all network participants share the same ledger [22]. All transaction data in the network are included in blocks,

and the blocks are linked to each other in a chain form. Blockchain is a decentralized method in which nodes in the network each own a ledger, rather than a method in which a central server manages data. Therefore, as the nodes in the network directly verify the transaction, the transaction is performed without a third party and does not require a server. If hackers want to manipulate data, they must manipulate the blockchains of the majority nodes in the network. However, since this is virtually impossible, the integrity of the transaction is guaranteed. Due to these advantages, blockchain is used in various fields, such as digital asset transactions and medical care.

2.5. Smart Contract

A smart contract is a digital contract based on blockchain [23]. In the past, contracts were made in writing, but smart contracts are implemented through code. Smart contracts are automatically executed only when certain conditions are met. Therefore, the decentralization pursued by the blockchain is realized by the two parties signing a contract through a smart contract without the involvement of a third-party certification authority. Smart contracts can be written in a variety of high-level programming languages, but are typically written in the Solidity language. Code written in a high-level programming language is converted to Ethereum bytecode through a compiler and is then deployed on the blockchain.The Ethereum bytecode consists of an opcode and a value. An opcode represents an instruction to be executed by a computer, and there are currently 140 opcodes in Ethereum. The deployed smart contract is executed through the Ethereum virtual machine (EVM). EVM provides an execution environment for smart contracts and executes smart contracts in the form of a bytecode. In addition, in order to execute a smart contract, it is necessary to have a certain amount of gas, which is used as a fee for the transaction. This is the cryptocurrency paid to miners as the computational cost of smart contract execution.

Once deployed, the smart contract cannot be deleted and its code cannot be modified. This is because smart contracts are deployed on a blockchain that has immutability characteristics. Information stored on the blockchain is permanently maintained and cannot be modified. In other words, even if errors or security vulnerabilities are found in the smart contract code, they cannot be corrected, which can cause big problems.

2.6. Greedy Contract

A greedy contract [4] is a smart contract that can reach a state where it locks indefinitely so that the Ether cannot be withdrawn. A greedy contract can receive the Ether. However, since there is no instruction to process the received Ether or the instruction cannot be reached, the Ether is locked in the contract forever. Therefore, the Ether sent to the greedy contract cannot be recovered even by the node that deployed the contract. Executing a greedy contract can cause enormous damage.

2.7. Malicious Smart Contract Detection

ACSAC '18 classified malicious smart contracts into three types: greedy, suicidal, and prodigal contracts [4]. In addition, the authors implemented `MAIAN`, a tool that detects malicious smart contracts through symbolic analysis. The `MAIAN` tool is the most representative smart contract detection tool. It detects bug-causing transactions by processing the bytecode of smart contracts. In the symbolic analysis method, the symbolic execution is started at the first instruction in the bytecode. Execution then proceeds sequentially until the terminating instruction (e.g., STOP, RETURN) is found. If a function call occurs while tracing an instruction sequence, it is searched considering the branch condition. If the terminating instruction is not valid, it is backtracked in the depth-first search process to try another path. Then, concrete validation is performed to verify the results of symbolic analysis. For this, malicious contract candidates are executed on a fake Ethereum network created through a private fork. In order to distinguish between the three types of malicious contracts, the following procedures are performed: Ether check (prodigal), check if the

contract can be killed (suicidal), instruction check (greedy), etc. If the contract is determined to be malicious even after this process, `MAIAN` finally classifies the contract as malicious.

The `MAIAN` tool is developed in Python, and it can detect a smart contract in less than 10 s on average.

In [5], the authors proposed a system that detects malicious contracts after first detecting malicious users. This is the latest work in the field of malicious smart contract detection. In this system, if a user deploys a malicious smart contract, the user is classified as a malicious node and can no longer deploy smart contracts. LSTM, GRU, and ANN were used as models to detect malicious smart contracts. The AUCs of LSTM, GRU, and ANN achieved 0.99, 0.99, and 0.97, respectively. In the scenario considered, detection is performed before deploying smart contracts, so detection is not possible for smart contracts that have already been deployed.

In [6], SmartCheck, which analyzes XPath patterns by converting Solidity codes into XML-based intermediate representations, was proposed. SmartCheck detects vulnerabilities in smart contracts by analyzing the XPath. However, a weak smart contract without an XPath pattern has a critical problem in that it is difficult to detect.

In [7], SNC '21 proposed a model that analyzes the vulnerability of smart contracts based on machine learning by introducing a shared child node. The model can predict eight types of vulnerabilities including re-entrancy, arithmetic, access control, unchecked low-level calls, bad randomness, front running, and denial of service. However, the model is not stable because it does not secure enough malicious smart contracts.

3. Greedeeptector

For the well-known `MAIAN` tool, various detection methods have been proposed. In previous work [5], the latest deep learning-based detection technique is described, which can also detect malicious contracts, but the number of datasets used is very small. Therefore, the reliability is not high, and it is difficult to state that it is robust. Also, the methods cannot perform detection on already deployed smart contracts.

In this paper, we present a robust greedy contract detection system using lightweight deep learning implemented through XAI. Our approach works both for contracts to be deployed and contracts that have already been deployed. Therefore, the proposed method can ensure the safe execution of smart contracts on the current Ethereum network. In addition, we design a lightweight detection system that can operate efficiently on the blockchain. Moreover, in this work, we evaluate and discuss our detection system. Our work is applied to the blockchain network. That is, the deep learning model is run on blockchain nodes (e.g., computers or low-end devices). Evaluation of the execution of the actual blockchain network will be carried out in future work.

Figure 1 and Algorithm 1 show the mechanism of `Greedeeptector`. nodes on the Ethereum network can obtain information about smart contracts and execute smart contracts. Before executing a smart contract, each node runs `Greedeeptector` to detect a greedy contract. For this, pre-processing is performed. As mentioned in line 2 of Algorithm 1, the smart contract's bytecodes are converted to opcodes (https://github.com/daejunpark/evm-disassembler accessed on 11 September 2023). Then the frequency for each opcode is counted. Finally, the opcode frequency of the target smart contract is input into `Greedeeptector`. If the result (*isGreedy*) is 0, then the contract is executed because it is a benign contract. Conversely, if it is a greedy contract, it is not executed. Table 1 represents the key points of the proposed system.

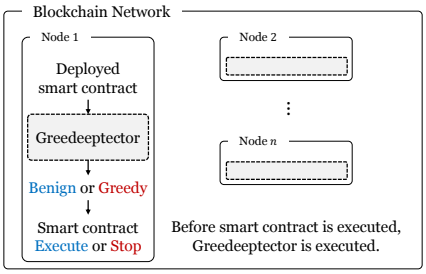

Figure 1. Diagram of our Greedeeptector.

Algorithm 1 Greedeeptector mechanism

Require: Bytecode of smart contract (B_{SC}), Extracted opcodes of smart contract (OP_{sc}), Frequency of opcodes (F_{OP}), Deep learning model for greedy contract detection (`Greedeeptector`)

Opcodes={00:STOP, ..., *FF*:SELFDESTRUCT} ▷ Set up dictionary mapping opcodes to bytecodes
for *op* in B_{SC} **do**
  if *op* in *Opcodes* **then**
    OP_{sc}.append(*op*) ▷ Bytecode to opcode
  end if
end for
Initialize F_{OP} to zero
for *op* in OP_{sc} **do**
  $F_{OP}[op] \leftarrow F_{OP}[op]+1$ ▷ Calculate frequency of opcode
end for
isGreedy ←`Greedeeptector` (F_{OP}) ▷ Greedy contract detection
if *isGreedy* == *True* **then**
  Stop the smart contract
else
  Execute the smart contract
end if

Table 1. Key points of the proposed system.

Key Points	Descriptions
Greedy contracts	Greedy contracts have potential risk (e.g., Ether can be lost.).
Blockchain network	Blockchain network can execute smart contract (e.g., Ethereum, Bitcoin).
Explainable artificial intelligence	Since it can interpret the results of training, it can be used for lightweighting.
Lightweight deep learning	It increases the scalability of blockchain and can be applied to lightweight devices.

3.1. Base Model

In this section, we present the base model of `Greedeeptector`. Figure 2 shows a diagram of the base model. First, the opcodes are extracted from the smart contract. The extracted opcodes are converted into an array representing the frequency of each opcode. The frequencies for each opcode are input to the neural network, and the neural network classifies greedy and benign contracts. If the target smart contract is classified as a greedy contract, its execution is stopped.

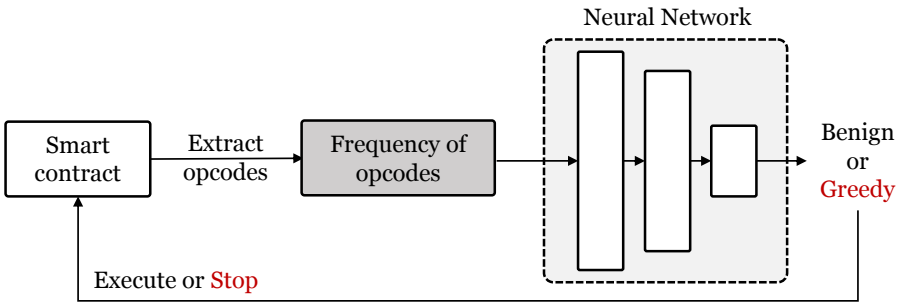

Figure 2. The diagram of the base model.

3.1.1. Pre-Processing

As explained earlier, the frequency of smart contract opcodes is required to detect greedy contracts. Therefore, the frequency of the opcode must be extracted from the bytecode of the smart contract through pre-processing. Figure 3 shows the pre-processing process. A smart contract can be expressed as a bytecode composed of opcodes (instructions) and values used for operation. We extract the opcodes among them. Then, a sequence of opcodes used in the smart contract is generated. Since one opcode is 1 byte, a total of 256 opcodes (00 to *FF*) can exist. However, Ethereum only provides 140 opcodes. So, we generate an array of length 140 representing the opcode frequency. The index of the frequency array means each opcode and the value means the frequency. That is, it contains information about how many times each opcode is used in a smart contract. The generated array of the opcode frequency is used as a dataset for greedy contract detection.

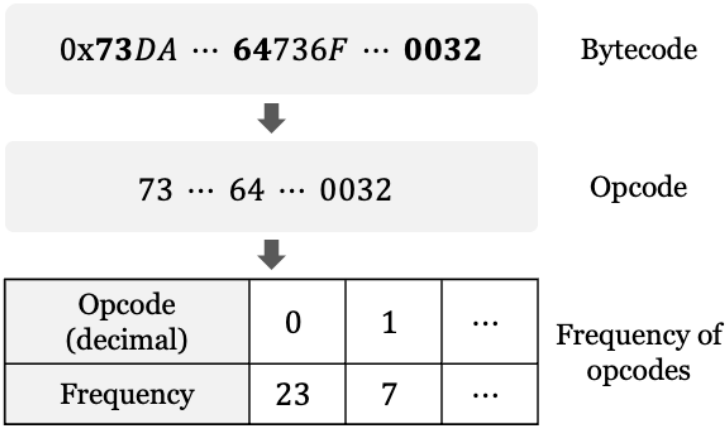

Figure 3. Converting bytecode to the frequency of opcodes in pre-processing.

3.1.2. Detection

The neural network is used to detect greedy contracts. Our proposed system is designed to perform detection without compromising blockchain scalability. Due to the large size of time-series models, like recurrent neural networks, we opted for a more lightweight deep neural network. Figure 4 shows the structure of the deep learning network used in our work. First, the opcode frequency data generated through the pre-processing is input to the deep learning network. Each element of the frequency array is assigned to one neuron of the input layer. That is, the number of neurons in the input layer is equal to the number of opcodes in Ethereum. As a hidden layer, a fully connected layer (linear layer) is used. Before passing to the output layer, it goes through a dropout layer that discards random neurons to prevent overfitting. In the output layer, a value between 0 (benign) and 1 (greedy) is output as a predicted value for the data through the sigmoid activation function. Equation (1) shows the formula for the sigmoid activation function. Finally, the loss between the actual label and the predicted value is calculated through the

binary cross-entropy loss function. Equation (2) represents the binary cross-entropy loss formula. y hat means the continuous sigmoid function output value between 0 and 1, and y means the discontinuous actual value. The neural network is updated to minimize the loss. Through this training process, a neural network can distinguish greedy contracts from benign contracts.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1}$$

$$BCE(y, \hat{y}) = -(y \log(\hat{y}) + (1-y) \log(1-\hat{y})) \tag{2}$$

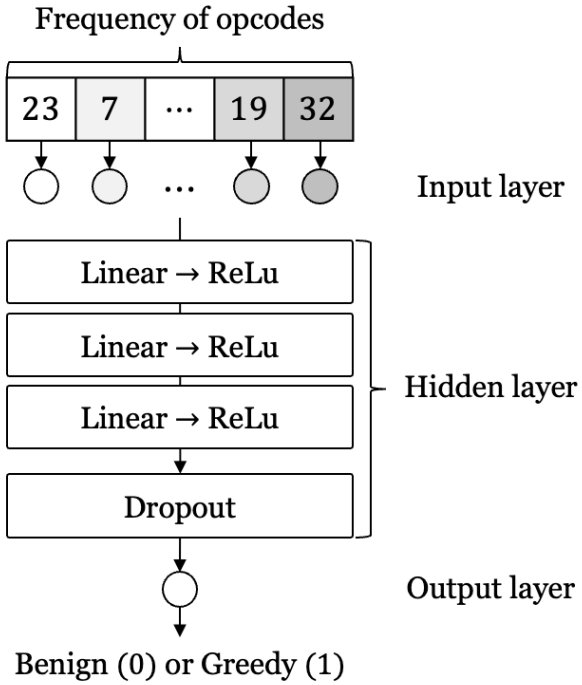

Figure 4. The structure of the base model.

Table 2 shows the hyperparameters of the base model. The hyperparameters of the base model are chosen through experimentation to achieve the optimal performance. To the best of our knowledge, these hyperparameters yield the maximum performance. The number of input layers is 140 (the number of opcodes used). Also, to prevent overfitting, a dropout value of 40% is used, and the epoch is set to 200.

Table 2. Hyperparameters of the base model.

Hyperparameters	Descriptions
Epoch	200
Batch size	256
Units of the input layer	140
Dropout	0.4
Optimizer (learning rate)	Adam (0.0001)

3.2. Lightweight Model Using XAI

We used XAI to design a computation and memory-efficient lightweight model. Figure 5 shows the process of implementing a lightweight model using XAI. We implemented the base model using the frequency of 140 opcodes. Subsequently, we computed the IG and SHAP values of the base model. In Figure 5, the red color in the graph represents the IG values, while the blue represents the SHAP values. Even though there are values for all 140 opcodes, the graph only captures 17 of them. These values indicate the influence

each feature of the input data has on the prediction; the larger their absolute value, the more significant the feature becomes. Furthermore, a positive value suggests that the feature contributes to the greedy contract, whereas a negative value implies a contribution to the benign contract. Thus, we can identify opcodes that have a significant impact on the classification between benign and greedy contracts.

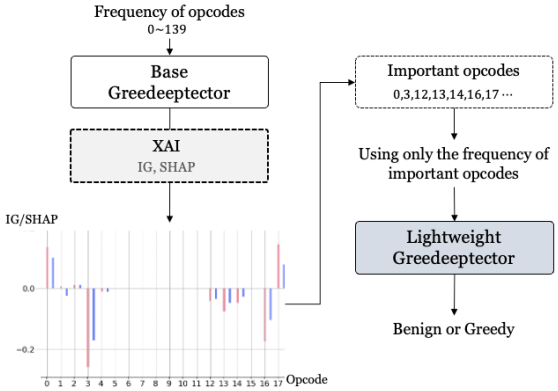

Figure 5. Design of lightweight model; the red color in the graph represents the IG values, while the blue represents the SHAP values.

The lightweight model uses only important opcodes among 140 opcodes. That is, fewer than 140 opcodes can be used. This usually allows the construction of a simpler model, since the number of input data features is reduced. As a result, the number of neurons in the input layer is reduced and the parameters of the neural network are reduced. Here, only opcodes that can reduce the parameters of the model as much as possible without degrading the accuracy should be selected. So, we chose the top n opcodes on the basis of experiment.

However, since a large number of data samples are used for training, some of them may have outliers. If there are outliers, the IG and SHAP values of a specific sample have a large effect on the average. Furthermore, the more positive the IG and SHAP values, the higher the contribution to greedy contracts, whereas the more negative the values indicates a higher contribution to benign contracts. The reliability is then reduced, so outliers should be removed. We use the inter-quantile range (IQR) [24] to remove outliers. In the inter-quartile range (IQR) approach, the dataset is segmented into quartiles, with the 25th percentile denoted as $Q1$ and the 75th percentile as $Q3$. The lower bound is established at $Q1 - 1.5 \cdot IQR$ and the upper bound at $Q3 + 1.5 \cdot IQR$. Any data points falling outside this interval are deemed outliers. The constant multiplied by IQR is usually 1.5. In summary, after removing outliers, our lightweight model can reduce the number of parameters while maintaining accuracy by learning only the top n opcodes from the average IG and SHAP.

Table 3 shows the hyperparameters of the lightweight model designed via XAI. The hyperparameters of the lightweight model are chosen through experimentation to achieve optimal performance. To the best of our knowledge, these hyperparameters yield the maximum performance. In the lightweight model, the same epoch, batch size, dropout rate, optimizer, and learning rate as the base model were used. In other words, the difference in hyperparameters between the lightweight model and the base model is the number of neurons in the input and hidden layers. In this work, the frequency of one opcode is assigned to one neuron. In the lightweight model, the number of units in the input layer is reduced because only the frequency for important opcodes is used as data (it is set experimentally, see Section 4.3). That is, since the number of features of the input data (the number of important opcodes) is reduced, the number of neurons in the model is reduced. As a result, lightweight models can reduce file size and computational complexity.

Table 3. Hyperparameters of the lightweight model.

Hyperparameters	Descriptions
Epoch	200
Batch size	256
Units of the input layer	58
Dropout	0.4
Optimizer (learning rate)	Adam (0.0001)

4. Experiments and Evaluation

For this experiment, we use Google Colaboratory Pro+, a cloud-based service with Ubuntu 20.04.5 LTS and GPU (Tesla T4) 15GB RAM. Python 3.9.16 and PyTorch 2.0.0 are used as the programming environment.

4.1. Dataset

In this experiment, the Google BigQuery dataset (https://cloud.google.com/blog/products/data-analytics/ethereum-bigquery-public-dataset-smart-contract-analytics?hl=en accessed on 11 September 2023) provided by Google is used. The Google BigQuery dataset contains datasets for smart contracts deployed on Ethereum. A dataset of 14,716 benign and greedy contracts is generated through MAIAN, a malicious smart contract detection tool. The training dataset includes both benign contracts and greedy contracts. Additionally, the test dataset also encompasses both benign and greedy contracts. The ratio of benign to greedy contracts is set at 1:1.

4.2. Result for Base Model

Table 4 shows the performance of the base model using 140 Ethereum instructions. The base model achieves an average test F1-score of 92.6%. The number of parameters of the base model is 15,297.

Table 4. Performance of the base model.

Performance Metric	Descriptions
Training F1-score	95.0%
Validation F1-score	92.3%
Test F1-score	92.6%
The number of parameters	15,297

4.3. Result for Lightweight Model

We use XAI to design a lightweight model that is computation- and memory-efficient. The lightweight model consists of a process of extracting important opcodes, removing outliers, and learning the frequency of important opcodes. In this section, we discuss the implementation and performance of the lightweight model.

Extract the Important Instructions Using XAI

Before selecting important opcodes, we need to remove outliers in IG and SHAP. This is because it improves the reliability of selecting important opcodes by reducing the impact of specific data samples on average for IG and SHAP. Figure 6 is an example of removing outliers using the IQR method described above. Most of the values are clustered between -1 and 1, and there are a few outliers. As shown on the right side in Figure 6, we successfully removed outliers.

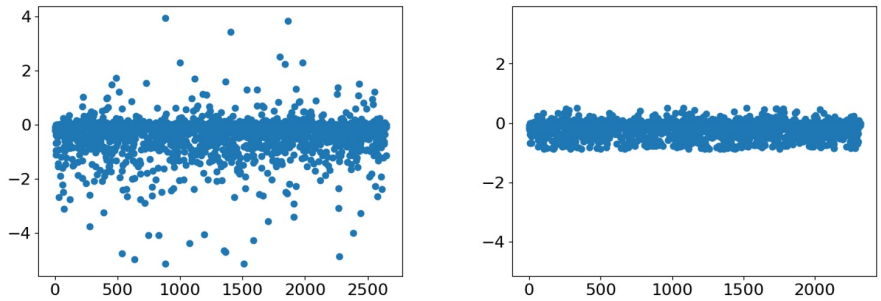

Figure 6. An example of outlier removal using the IQR ; Before (**left**), After (**right**).

The important instructions are selected by calculating IG and SHAP for the base model. The reason for considering both properties is to secure reliability. A total of 50 opcodes are selected from IG and SHAP to reduce the size of the model as much as possible while maintaining accuracy. This is the optimal number of opcodes obtained as a result of experimentation while reducing the number of instructions from 140 opcodes. Then, duplicates are removed from the 50 opcodes selected from IG and SHAP, respectively. This makes it possible to select important opcodes with both IG and SHAP.

However, the important opcodes are slightly different in each experiment. Therefore, it is difficult to obtain stable performance when using important opcodes obtained through one experiment. Therefore, we construct a lightweight model using instructions that appeared more than k times (k = 5, 7, 10) out of 10 trials.

Table 5 shows the experimental results for each case. The experimental findings indicate that the F1-score in `Case1` achieved the best accuracy with 92.3%. In `Case2` and `Case3`, the performance was lower than `Case1`. This seems to be because some opcodes that affect prediction are excluded.

Table 5. Comparison of performance according to the number of opcodes used (`Case1`, `Case2`, `Case3` ($k = 5, 7, 10$, respectively)).

	Case1	**Case2**	**Case3**
The number of opcodes	58	51	37
Training F1-score	0.93	0.92	0.90
Validation F1-score	0.92	0.92	0.90
Test F1-score	0.92	0.91	0.90
The number of parameters	5855	4861	3167

As the number of opcodes used decreases, the number of neurons in the input layer of the neural network decreases. Accordingly, the number of parameters in the entire model is also reduced. Therefore, `Case3` has the smallest number of parameters. However, `Case3` is unsuitable for use because of the large loss of accuracy compared to the base model. Therefore, we adopted `Case1`, which is lightweight with minimal loss of accuracy, as our `Lightweight Greedeeptector` model.

Table 6 shows the performance of the lightweight model. As noted earlier, the lightweight model has fewer parameters than the base model. Therefore, it is efficient in terms of computational complexity and memory usage. However, performance loss may occur. Nevertheless, the accuracy of the lightweight model was 92.3% So, our lightweight model provides memory and computational efficiency without loss of performance. Therefore, our work has the advantage of reducing computational and memory overhead in a blockchain network when blockchain nodes perform greedy contract detection in a real scenario.

Table 6. Performance of the lightweight model.

Performance Metric	Descriptions
Training F1-score	92.6%
Validation F1-score	91.6%
Test F1-score	92.3%
The number of parameters	5855

4.4. Comparison between Base Model and Lightweight Model

In this section, we report various experiments undertaken to build a lightweight model. We also compare the model size and detection accuracy of the base and lightweight models.

4.4.1. Performance

Table 7 shows a comparison between the base and lightweight model. The lightweight model size is reduced by 41.5%, and the space complexity decreases as the model size decreases. The parameter (weight of the model) is reduced by 61.8%, and the computational complexity decreases as the model has fewer parameters. The model size is a very important advantage in lightweight deep-learning models. A small-size model is equivalent to a deep-learning model with fewer parameters. The fewer the parameters, the smaller the amount of computation. Therefore, a small model size means low computational complexity. Thus, when detecting greedy contracts on the blockchain, our implementation is memory and computationally efficient. The accuracy of the base model and the lightweight model are similar. However, since we reduced the size of the model by 41.5%, this small performance loss is negligible.

Table 7. Comparison between base and lightweight model.

Model	Model Size	Parameters	Speed	F1-Score
Base	0.89 MB	15,297	0.015 ms	92.6%
Lightweight	0.53 MB	5855	0.013 ms	92.3%

In our scenario, our implementation is deployed on each node (IoT device) on the blockchain (currently, it is a prototype.). The efficiency of speed and memory is very important for low-end devices, such as IoT devices. In terms of blockchain scalability, the lower the computational and memory usage, the higher the scalability. In other words, our target node is a low-power node on the blockchain, so the small-size model is a significant advantage on the blockchain.

4.4.2. Instruction Analysis

Tables 8 and 9 show a comparison of the top eight important opcodes for benign and greedy contracts depending on the XAI algorithm. In IG and SHAP, it stands out that instructions related to branching (JUMP, JUMPDEST, JUMPI) are important. JUMP is an unconditional branch instruction, and JUMPI is a conditional branch instruction. JUMPDEST means the branch destination address. The branch instructions related to the benign contract are JUMP and JUMPDEST. Conversely, the branching instruction associated with greedy contracts is JUMPI. JUMPI is a conditional branch instruction, and JUMP and JUMPDEST are instructions that branch regardless of conditions. Due to greedy contract characteristics, the greedy contract uses conditional branch instructions to bypass the ability to process the Ether [4]. In fact, through our experiments, it is confirmed that the JUMPI instruction is an important instruction—it is thought that the JUMPI instruction greatly affects classification as a greedy contract. Conversely, in benign contracts, conditional branch instructions are not an important characteristic.

Table 8. Top 8 important opcodes in the benign contract.

Algorithm	Sorted by Values of IG and SHAP							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
IG	JUMPDEST	DUP1	SUB	JUMP	EQ	PUSH4	REVERT	SLT
SHAP	DUP1	SUB	JUMP	JUMPDEST	PUSH4	EQ	MLOAD	LT

Table 9. Top 8 important opcodes in the greedy contract.

Algorithm	Sorted by Values of IG and SHAP							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
IG	JUMPI	CALLVALUE	SWAP1	SWAP2	STOP	CALLDATASIZE	AND	ADDRESS
SHAP	JUMPI	PUSH2	CALLVALUE	SWAP2	POP	STOP	SWAP1	CALLDATASIZE

4.5. Comparison with Existing Method

`MAIAN` detects prodigal, suicidal and greedy contracts. Its file size is very small. However, the disadvantage is that detection takes a long time.

The previous work represents the latest deep-learning-based approach. It categorizes several types of malicious smart contracts into one class. Therefore, various malicious smart contracts can be detected. However, the methods used do not work on blockchain networks and are not valid for already deployed smart contracts. In other words, it is operable only for newly deployed smart contracts. Therefore, it is not suitable for blockchain networks where numerous malicious contracts have already been deployed. In addition, data from a total of 781 contracts (650 benign contracts and 131 malicious contracts) only were used. Therefore, it is considered that the reliability of the previous approach is not high. The elements mentioned in Table 10 are the result of our implementation of the previous model as it is.

Many models have been proposed in previous work, but we reproduce the model with the highest performance. These values are then measured by the reproduced model.

Table 10. Comparison of smart contract detection methods.

Category	**MAIAN** [4]	Previous Work [5]	This Work (**LM**)
Smart contracts	Prodigal, suicidal, greedy (3 classes)	Malicious (2 classes)	Greedy (2 classes)
Algorithm	Symbolic analysis, Concrete validation	Deep learning	Deep learning
Parameters	-	54,018	5855
File size	0.232 MB	0.74 MB	0.53 MB
Speed	Within 10 s	Slower than ours	0.013 ms

Our method can classify greedy contracts and benign contracts. However, among the malicious contracts that occur when using a dataset of about 80,000, the proportion of greedy contracts is 93.4% (in the Google BigQuery dataset). Therefore, problems caused by malicious smart contracts in the blockchain can be prevented just by detecting greedy contracts. We increased the reliability and stability of the model by using a dataset 38 times larger than in previous work. Our implementation only detects greedy contracts but has the potential to prevent most malicious smart contracts. Our lightweight model is larger than the `MAIAN`, but can detect contracts significantly faster. Compared to the previous work, the file size and the number of parameters decreased by 28.4% and 89.2%, respectively. Since our work has far fewer parameters, it is fast and memory efficient. These points represent a great advantage for the blockchain network.

5. Conclusions

In this work, we identify and analyze opcodes that significantly impact the model in order to implement a lightweight model. Additionally, based on this, we propose a computationally and memory-efficient lightweight detection model for the IoT. Unlike previous

approaches that perform detection at the smart contract deployment phase, the proposed system performs detection at the smart contract execution phase. Therefore, the proposed system can perform the detection of already deployed smart contracts. As a result, the proposed system improves the stability and scalability of blockchain networks.

As a summary of our experimental results, the file size of the lightweight model is reduced by 41.5% compared to the base model. Thus, when our system is running on a blockchain, it is a memory and computationally efficient system. Finally, despite being lightweight, our system has a high detection accuracy of 92.3%.

We tried to detect not only greedy contracts but also various malicious smart contracts, but it is not easy to collect malicious smart contract datasets other than greedy contracts. Therefore, we implemented a model that detects only greedy contracts.

In future work, we plan to implement a model that can detect various malicious smart contracts as well as greedy contracts. Additionally, the lightweight model is implemented to target low-end devices. Therefore, we plan to make the model lighter by using not only XAI but also knowledge distillation, quantization, and pruning methods. Afterwards, we plan to conduct additional experiments by deploying the lightweight model on low-end devices, such as Raspberry Pi.

Author Contributions: Software, Y.K., W.K. and H.K.; Writing—original draft, Y.K.; Writing—review & editing, M.L., M.S. and H.S.; Supervision, H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00627, Development of Lightweight BIoT technology for Highly Constrained Devices, 100%).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liu, Z.; Jiang, L.; Osmani, M.; Demian, P. Building information management (BIM) and blockchain (BC) for sustainable building design information management framework. *Electronics* **2019**, *8*, 724. [CrossRef]
2. Liu, J.; Liu, Z.; Yang, Q.; Osmani, M.; Demian, P. A Conceptual Blockchain Enhanced Information Model of Product Service Systems Framework for Sustainable Furniture. *Buildings* **2022**, *13*, 85. [CrossRef]
3. Liu, Z.; Wu, T.; Wang, F.; Osmani, M.; Demian, P. Blockchain Enhanced Construction Waste Information Management: A Conceptual Framework. *Sustainability* **2022**, *14*, 12145. [CrossRef]
4. Nikolić, I.; Kolluri, A.; Sergey, I.; Saxena, P.; Hobor, A. Finding the greedy, prodigal, and suicidal contracts at scale. In Proceedings of the 34th Annual Computer Security Applications Conference, San Juan, PR, USA, 3–7 December 2018; pp. 653–663.
5. Shah, H.; Shah, D.; Jadav, N.K.; Gupta, R.; Tanwar, S.; Alfarraj, O.; Tolba, A.; Raboaca, M.S.; Marina, V. Deep Learning-Based Malicious Smart Contract and Intrusion Detection System for IoT Environment. *Mathematics* **2023**, *11*, 418. [CrossRef]
6. Tikhomirov, S.; Voskresenskaya, E.; Ivanitskiy, I.; Takhaviev, R.; Marchenko, E.; Alexandrov, Y. Smartcheck: Static analysis of ethereum smart contracts. In Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, Gothenburg, Sweden, 27 May 2018; pp. 9–16.
7. Xu, Y.; Hu, G.; You, L.; Cao, C. A novel machine learning-based analysis model for smart contract vulnerability. *Secur. Commun. Networks* **2021**, *2021*, 1–12. [CrossRef]
8. Lohith, J.J.; Anusree Manoj, K.; Guru, N.; Srinivasan, P. TP-Detect: Trigram-pixel based vulnerability detection for Ethereum smart contracts. *Multimed. Tools Appl.* **2023**, 1–15. [CrossRef]
9. Tyagi, A.K.; Dananjayan, S.; Agarwal, D.; Thariq Ahmed, H.F. Blockchain—Internet of Things Applications: Opportunities and Challenges for Industry 4.0 and Society 5.0. *Sensors* **2023**, *23*, 947. [CrossRef] [PubMed]
10. Taloba, A.I.; Elhadad, A.; Rayan, A.; Abd El-Aziz, R.M.; Salem, M.; Alzahrani, A.A.; Alharithi, F.S.; Park, C. A blockchain-based hybrid platform for multimedia data processing in IoT-Healthcare. *Alex. Eng. J.* **2023**, *65*, 263–274. [CrossRef]
11. Sharma, P.; Namasudra, S.; Crespo, R.G.; Parra-Fuente, J.; Trivedi, M.C. EHDHE: Enhancing security of healthcare documents in IoT-enabled digital healthcare ecosystems using blockchain. *Inf. Sci.* **2023**, *629*, 703–718. [CrossRef]
12. Haykin, S. *Neural Networks and Learning Machines, 3/E*; Pearson Education: Chennai, Tamil Nadu, India, 2009.
13. Albawi, S.; Mohammed, T.A.; Al-Zawi, S. Understanding of a convolutional neural network. In Proceedings of the 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 21–23 August 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6.
14. Petneházi, G. Recurrent neural networks for time series forecasting. *arXiv* **2019**, arXiv:1901.00069.

15. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial networks. *Commun. ACM* **2020**, *63*, 139–144. [CrossRef]
16. Wang, C.H.; Huang, K.Y.; Yao, Y.; Chen, J.C.; Shuai, H.H.; Cheng, W.H. Lightweight deep learning: An overview. *IEEE Consum. Electron. Mag.* **2022**, 1–12. [CrossRef]
17. Gunning, D.; Stefik, M.; Choi, J.; Miller, T.; Stumpf, S.; Yang, G.Z. XAI—Explainable artificial intelligence. *Sci. Robot.* **2019**, *4*, eaay7120. [CrossRef] [PubMed]
18. Ali, S.; Abuhmed, T.; El-Sappagh, S.; Muhammad, K.; Alonso-Moral, J.M.; Confalonieri, R.; Guidotti, R.; Del Ser, J.; Díaz-Rodríguez, N.; Herrera, F. Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence. *Inf. Fusion* **2023**, *99*, 101805.
19. Kokhlikyan, N.; Miglani, V.; Martin, M.; Wang, E.; Alsallakh, B.; Reynolds, J.; Melnikov, A.; Kliushkina, N.; Araya, C.; Yan, S.; et al. Captum: A unified and generic model interpretability library for pytorch. *arXiv* **2020**, arXiv:2009.07896.
20. Sundararajan, M.; Taly, A.; Yan, Q. Axiomatic attribution for deep networks. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 3319–3328.
21. Erion, G.; Janizek, J.D.; Sturmfels, P.; Lundberg, S.M.; Lee, S.I. Learning explainable models using attribution priors. In Proceedings of the International Conference on Learning Representations ICLR 2020, Addis Ababa, Ethiopia, 30 April 2020.
22. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Bus. Rev.* **2008**, *21260*, 1–9.
23. Buterin, V. A next-generation smart contract and decentralized application platform. *White Pap.* **2014**, *3*, 1–2.
24. Whaley, D.L., III. The Interquartile Range: Theory and Estimation. Ph.D. Thesis, East Tennessee State University, Johnson City, TN, USA, 2005.