IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 6 July 2023, accepted 21 July 2023, date of publication 25 July 2023, date of current version 1 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3298672

METHODS

Improvement and Optimization of Vulnerability Detection Methods for Ethernet Smart Contracts

ZHONGJU YANG, WEIXING ZHU, AND MINGGANG YU
Command and Control Engineering College, Army Engineering University of PLA, Nanjing, Jiangsu 210000, China
Corresponding author: Weixing Zhu (zwx@aeu.edu.cn)

This work was supported by project (No. KYZYJKKCJC23001).

ABSTRACT Smart contracts based on blockchain are widely used in finance, management, Internet of Things, healthcare, and other fields. However, with the rapid development of smart contracts, the corresponding security vulnerability attack cases occur frequently. Existing Ethereum smart contract vulnerability detection tools based on static analysis techniques rely too much on expert rules, for this reason, this paper proposes an Ethereum smart contract vulnerability detection method SCSVM based on support vector machine technology. A representation of smart contracts is constructed based on the word-to-vector technique, the features of Ethereum smart contracts are extracted based on the support vector machine technique, and these features are combined to identify vulnerabilities. Experiments on Smartbugs and Smartbugs-wild show that SCSVM is significantly effective. It achieves a detection accuracy of 87.51%, outperforming five typical static analysis vulnerability detection tools in terms of F1-score. To alleviate the problems of deep learning methods over-relying on large-scale data to train models and collecting a large number of smart contract attack samples in a short period, this paper proposes a basic learner-meta-learner framework, SCLMF. solc-based acquisition of the bytecode of Ethereum smart contract Solidity, on which smart contract representations are constructed via Python and the use of SCLMF for vulnerability detection. The experiments on WScrawlD show that SCLMF has a certain detection effect. Also, to further verify the effectiveness of SCLMF, experiments were conducted on Omniglot, and the detection accuracy was 96.7% and 98.5% under 5-way 1-shot and 5-way 5-shot conditions, respectively, which exceeded Memory-Augmented Neural Networks and CONVOLUTIONAL SIAMESE NETS. In summary, the experiments proved the effectiveness of SCSVM and SCLMF in Ethereum smart contract vulnerability detection.

INDEX TERMS Base learner-meta-learner, Ethereum, smart contracts, support vector machines, vulnerability detection, word embedding.

I. INTRODUCTION

Cryptographer Nick Szabo first introduced the term "smart contracts" to describe the automation of ordinary legal contracts in the 1990s; specifically, contracts that utilize computer language to record terms and are automatically performed by a program [1]. However, the application and development of smart contracts were once restricted by the lack of a trusted execution environment. It wasn't until the advent of blockchain that a reliable environment for smart contracts enabled for their efficient application on the technology. The open and transparent, unchangeable, and perpetual operation of blockchain data are properties of smart contracts that have been deployed [2]. At the same time, when the predefined requirements are satisfied, the contract's provisions, which were written in a computer program, are automatically carried out, and the entire process is independent of a third party. Smart contract implementation strengthens the decentralized character of blockchain platforms and boosts use cases for blockchain, including the Internet of Things, banking, and healthcare [3]. However, because smart contracts oversee significant assets like digital currencies, attackers are highly motivated to target smart contracts in order to forcibly acquire and hold onto

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

digital currency holdings. In addition to causing enormous financial losses, smart contract security events also jeopardize the blockchain-based credit system. To keep the blockchain secure, it is necessary to find insecure contracts before they are distributed, as smart contracts are difficult to change once they are placed on the blockchain.It is important to research the vulnerability detection of Ethernet smart contracts since, among them, Ethernet is the most widely used blockchain platform for running smart contracts.

Deep learning and machine learning are currently utilized frequently and have advanced significantly in many industries, although they still have limits. Both rely on large-scale data training to accomplish the target task but often fail to achieve satisfactory results for real-world situations with small sample sizes or annotated samples [4], [5]. However, in the field of Ethereum smart contract vulnerability detection, the sample size of real-world vulnerable contracts is small, and it is difficult for security agencies to collect sufficient vulnerability samples in a short period. For this reason, it is important to study the detection of Ethereum smart contract vulnerabilities under small samples. The study of Ethernet smart contract vulnerability detection is concerned with the security of transactions on the Ethernet platform and reducing the economic losses arising from vulnerable contracts. Machine learning and deep learning have played an important role in the field of Ethernet smart contract vulnerabilities, with good results in terms of detection speed and detection accuracy. However, there are few examples of Ethereum smart contract vulnerabilities in the real world, making it challenging for security agencies to gather enough examples in a short amount of time. As a result, researching small-sample learning techniques becomes a crucial step in finding a solution and overcoming other obstacles in the field of identifying smart contract vulnerabilities.

To this end, this paper improves and optimizes the current methods for detecting vulnerabilities in Ethernet smart contracts, and the main contributions include the following three topics. First, we study machine learning for the detection of vulnerabilities in Ethernet smart contracts. In order to improve the accuracy of Ether smart contract vulnerability detection, we propose an SCSVM method based on Word2Vec (a word embedding technique) and SVM (a machine learning method). In addition, the SCSVM method can alleviate the problem that current static analysis tools for detecting smart contract vulnerabilities rely too much on expert rules resulting in the inability to reuse rules among different vulnerability types. The second one is based on the open-source ScrawlD [7] and Smartbugs-wild [6] datasets and the Python programming language to create the WScrawlD Ethereum smart contract image dataset. This can help Ether security agencies to conduct experimental research on vulnerability identification more effectively. The third is to study the vulnerability detection of Ethernet smart contracts with fewer samples. By introducing meta-learning into Ether smart contract vulnerability detection through the SCLMF method, which is based on the learner-meta-learner framework, we hope to address the problem of deep learning models over-relying on big data. Through experiments, we show that our proposed SCSVM technique has considerable benefits and practical value for identifying vulnerabilities in Ethernet smart contracts. the SCLMF method is the first attempt to incorporate meta-learning into Ethernet smart contract vulnerability detection and provides an important reference value for future research on Ethernet smart contract vulnerability detection methods under fewer sample conditions.The research of SCSVM method and SCLMF method is a complementary research, which can effectively adapt to the detection of vulnerabilities of Ethernet smart contracts in different scenarios and has great practical significance.

Therefore, this study focuses on the problem of Ethernet smart contract vulnerability detection and proposes a series of improvement and optimization methods. We propose the SCSVM method, which has been experimentally validated on the publicly available datasets Smartbugs-wild and ScrawlD. The results demonstrate that the SCSVM method has a significant performance advantage in the detection of Ethernet smart contract vulnerabilities. Additionally, we suggest the SCLMF method based on small-sample learning techniques, which employs meta-learning to address the issue of deep learning models overly depending on large-scale data, allowing the identification of small-sample Ethereum smart contract vulnerabilities. The contribution of this study will make an important contribution to improving the security of transactions on the Ether platform, safeguarding the functionality that smart contracts were originally designed to achieve, and maintaining a blockchain-based credit system.

The research of the SCSVM method and SCLMF method is generally complementary. The SCSVM method focuses on alleviating the problem that current static analysis tools rely too much on expert rules when detecting vulnerable contracts and it is difficult to reuse rules among different vulnerability types; the SCLMF method focuses on alleviating the problem that deep learning methods rely on large-scale data when detecting vulnerable contracts. This complementary research can effectively adapt to different scenarios of Ethernet smart contract vulnerability detection, and different solutions of SCSVM and SCLMF methods can be selected according to different Ethernet smart contract vulnerability detection needs. In addition, as the SCSVM method and SCLMF method are research works for different realistic problems in the field of Ethernet smart contract vulnerability detection, they should be relatively more effective in solving the corresponding problems. The selection and application of the two methods are more relevant in the case of different Ethernet smart contract vulnerability detection needs.

This work is organized as follows: in Section II, we describe research on vulnerability identification and small-sample learning for Ethernet smart contracts. We shall

go over our strategy in Section III. The dataset and execution performance comparison are included in Section IV of our presentation of the experimental approach. In Section V, we wrap up the entire essay.

II. RELATED WORK

A. WORD EMBEDDING TECHNIQUES

Word embedding is a technique for mapping words into vector space, which plays an important role in natural language processing. By representing words as low-dimensional vectors, word embeddings can better express the semantic relationships between words and thus achieve efficient and accurate results in NLP tasks such as text classification, sentiment analysis, and machine translation. Commonly used word embedding generation techniques include Word2vec, GloVe, etc. They can be widely used in different fields, such as recommendation systems, cross-language translation, etc. In addition, word embeddings can be combined with other machine learning techniques and linguistic resources to further improve model performance, such as combining knowledge bases and corpora to generate better word embedding models. Singular value decomposition (SVD) and other methods have been used in several research to minimize the dimensionality of sparse word-context matrices [8]. Word2vec refers to two language models that produce dense vector representations of words based on the Mikolov et al. neural network [9].

To calculate dense vector representations of words from extremely large datasets continuous vector representations of words, Mikolov et al. offer two novel model designs. In a word similarity task, the effectiveness of these representations was evaluated, and the findings were contrasted with those obtained from earlier best representation methods based on various types of neural networks. It was demonstrated that there was a significant increase in accuracy at a much reduced computational cost. In contrast to the prediction model Word2vec, Peng et al.'s Global Vectors for Word Representation (GloVe) decreases the dimensionality of the co-occurrence matrix of word-word types produced by a fixed-dimensional local context window. The name GloVe comes from the fact that the model directly captures the statistics of the entire corpus (at the global level) [10]. Additionally, it performs better and is more competitive than other cutting-edge techniques in tasks including named entity identification, word analogies, and word similarity. One of the main applications of word embeddings is the semantic similarity evaluation of words in several languages, which essentially dates back to the first application stage of natural language processing. In this sense, the paper [11] proposes a model called Bi-lingual Word Embedding Word Skipping (BWESG), which introduces a multilingual vector space to embed word representations, queries, and even complete documents, to jointly learn bilingual embeddings based only on comparable data consisting of aligned documents in two different languages. In a similar vein, Glavas et al. [12] suggest a different method for comparing the textual semantic similarity of documents written in various languages. This method uses fewer resources and is represented by a linear transfer of words from the vector space to the language of the vector space's language origin. GloVe and CBOW were utilized to create the word embeddings used in this study. Word embeddings have also been suggested for languages like Arabic that use extremely particular alphabets. To give the community access to word embeddings produced from various domains, such as Arabic tweets, websites, and Wikipedia articles, Soliman et al. [13] proposed a collection of pre-trained word representation models for Arabic. Word embeddings in Arabic have also been suggested as a way to solve the problem of word disambiguation, a frequent task in natural language processing.

Specifically, Laatar et al. [14] suggested using this approach to create a dictionary that illustrates the development of the meaning and usage of Arabic words, which would help to preserve the Arabic cultural heritage. The word embedding generation technique involved is the Word2vec architecture. Word embeddings can be merged into recommender systems and Musto et al. [15] present a preliminary investigation employing word embeddings where both objects and user profiles are embedded in a vector space for use in content-based recommender systems.According to Greenstein et al. [16], it is possible to translate the user's desired item sequence into words so that it can be projected into a vector space where parallels and similarities between objects can be found. The Word2vec and GloVe models are utilized to generate the word embeddings. Word embeddings may also be utilized in conjunction with other machine learning methods or linguistic data. According to the paper [17], methods that produce vector representations of words solely based on data dispersed throughout the corpus do not take advantage of the structure of semantic relations between words in concurrent contexts; these structures are intricate knowledge bases like ontologies and semantic vocabularies in which the meaning of words is defined by the various relations that exist between them. As a result, when utilizing word embeddings to produce findings that support the premise, integrating the corpus and knowledge base can enhance performance on word similarity and analogies tasks. Liu [18] suggested that in addition to creating vector representations of words using the corpus as the source, intrinsic word components such morphemes should also be taken into account. The morphology of the original view and morphology of the contextual view (MOMC) and morphology of the contextual view (MC), which exceed baseline models in detecting word similarity, including CBOW, are presented as two models for creating word embeddings to achieve this goal.A method to embed Word2vec-generated word embeddings into photos and then use a convolutional neural network (CNN) to classify the images as text was proposed by Gallo et al. [19]. Comparing the approach's classification results to baseline values (doc2vec vs. SVM), the method produced better results.

B. ETHERNET SMART CONTRACT VULNERABILITY DETECTION

Machine learning and deep learning-based detection methods significantly improve detection efficiency by automating feature extraction and proper model training for different forms of contract code, making them more generalizable and applicable to more application scenarios, and improving the detection accuracy of common vulnerabilities to a certain extent compared to existing tools. While dealing with the vulnerability detection problem, treating contracts as different objects has corresponding different solutions. It is noted in the paper that the data set used in the CBGRU model study was relatively small at the time, but the deep learning model can perform better detection on massive data sets. The CBGRU model proposed in the paper [4] has high accuracy in smart contract vulnerability detection tasks but still has limitations. The CBGRU model study demonstrates that the model's performance is constrained by its excessive reliance on data. A contract vulnerability detection approach called SCVDIE, which is based on seven different neural networks and employs contract vulnerability data for contract-level vulnerability detection, is proposed in the paper [5]. SCVDIE is different from general deep learning detection methods, SCVDIE can achieve better detection results with reduced dataset size, and it is pointed out in the paper that future research work considers applying migration learning in this area.SCVDIE can be regarded as the first exploration of Ethereum smart contract vulnerability detection in small samples, but it has not been studied in depth.

SVChecker [20] is a method that transforms the Ethereum smart contract vulnerability detection problem into a text classification problem processing, which consists of three main phases according to the core modules: code fragment extraction, deep learning model, and checker for the unknown source code of smart contract solidity. In the code extraction stage, irrelevant information from the contract source code is disregarded, the source code is program-sliced, and the resulting program slices are normalized, such that function names are consistently named FUN1-N and variable names are consistently named VAR1-N, to obtain code fragments and identify whether or not they are vulnerable. The code fragment is encoded into the input form (vector) of the neural network model using the transformer-encoder-based model in the deep learning model stage after the word embedding technique. Next, the fully connected layer transforms the high-dimensional vector into a low-dimensional vector, and the detection result is then output through the classification function. The smart contract solidity's source code is utilized as input, extracted using code fragments, and trained using a neural network model to produce the final classification result. This is what is meant by the checker for the unknown source code of the smart contract solidity. It is shown that this system has higher detection accuracy compared to the existing tools Oyente, Securify, Slither, and Smartcheck. CodeNet [21] is a transformation of the Ethernet smart contract vulnerability detection problem into an image classification and recognition problem processing, mainly to alleviate the problem of ignoring its semantics and context in the process of detecting smart contract vulnerabilities based on deep learning techniques. CodeNet implementation of smart contract vulnerability detection is mainly divided into two phases, which are the preprocessing phase and the detection phase. In the preprocessing phase, the source code of the Ethereum smart contract solidity is transformed into an image. The steps are as follows: firstly, compile the smart contract source code into bytecode, and secondly, convert the bytecode into an input image based on the smart contract, while preserving the semantics and context of the smart contract. In the detection phase, the smart contract-based input image is analyzed using the proposed CodeNet-based vulnerability detection method.

Peculiar [22] uses pre-training approaches to convert the Ethernet smart contract solidity source code into a non-Euclidean graph issue for processing and finds vulnerabilities in Ethernet smart contracts based on important data flow graphs. The key dataflow graph is less complex and does not include unnecessary deep hierarchies when compared to standard dataflow graphs that are currently employed in existing methodologies, which makes it simple for the model to concentrate on important properties. The model also includes pre-training techniques as a result of the significant advancements it has made in numerous NLP jobs. According to the empirical findings, the Peculiar technique can identify re-entry vulnerabilities in Ethereum smart contracts with an accuracy rate of 91.80 percent and a recall rate of 92.40 percent. This detection method greatly surpasses previous cutting-edge methods for detecting reentrant vulnerabilities, one of the most critical and prevalent Ethereum smart contract issues. A unified platform for developing smart contracts called SCStudio [23] seeks to make it simple for programmers to use more secure smart contracts. The main concept is to provide pattern-based learning-based real-time security-enhanced suggestions and integrated testing-based security-focused verification. scStudio is implemented as a VS code plug-in. It has been included as a suggested development tool by the FISCO-BCOS community and utilized as an official development tool for WeBank. In actual use, it surpasses currently available contract development environments like Remix, enhancing the average word suggestion accuracy by 30 to 60% and assisting in the discovery of roughly 25% of vulnerabilities. A multi-task learning-based vulnerability detection methodology for smart contracts is suggested in the study [24]. Setting auxiliary tasks to learn additional directional vulnerability traits enhances the model's detection power, allowing for the detection and identification of vulnerabilities. The model has two components and is based on a hard-share concept. First, the semantic information of the input contract is primarily learned from the underlying common layer. The feature vector of the contract is learned and extracted using an attention mechanism-based neural

network after the text representation is first turned into a new vector by word and location embedding. Second, the functionality of each task is primarily implemented using a task-specific layer.

C. FEW-SHOT LEARNING

In recent years, to alleviate the problems of insufficient generalization ability and over-reliance on large-scale data for training models in the application of deep learning in various fields, scholars have researched meta-learning. The ability of deep learning models to generalize is enhanced by meta-learning, which seeks to learn new knowledge using existing knowledge. The following five categories can be used to categorize meta-learning research techniques: Metric-based meta-learning techniques, meta-learning techniques based on initialized parameters with good generalizability, meta-learning techniques based on gradient optimizers, meta-learning techniques based on external memory units, and meta-learning techniques based on data augmentation [29]. The metric-based meta-learning can construct suitable distance metrics on the training dataset to model answers to practical questions, and the twin network proposed by Koch et al. [30] is a typical representative of the metric-based meta-learning methods aiming to solve the single-sample learning image classification problem. The twin network approach has shown far better performance compared to the more effective classifiers on existing Omniglot datasets.

The MAML model [31] is a typical example of initialized parameter-based meta-learning methods based on strong generalization, where the task data used in the process of optimizing the loss function and the network parameter gradient descent optimization process are separated to improve generalization capability. MAML has a wide range of application areas, such as classification, regression, and reinforcement learning, so MAML models are called model-independent meta-learning models. The meta-learner model proposed by Ravi et al. [32] is a typical representative of gradient optimizer-based meta-learning methods, aiming to solve the problem of training another learner neural network classifier in the case of small samples. Santoro et al.'s [33] memory enhancement model is a typical representative of meta-learning methods based on external memory units, aiming to solve the problem of relying on the existence of deep multilayer neural networks prone to overfitting by introducing external storage modules.By combining meta-learning techniques with GAN generation models, MetaGAN [34] achieves the process of differentiating true data from false data in order to find tighter decision boundaries for the model and further improve the feature extraction capability of the model.

In this paper, we build a text-based representation of Ethernet smart contracts using Word2Vec, a word embedding technology. We then convert the problem of detecting vulnerabilities in Ethernet smart contracts into a text classification problem and suggest a SCSVM framework. In this study, we build an image-based representation of Ethernet smart contracts based on bytecode and RGB image transformation technology, transform the small-sample Ethernet smart contract vulnerability detection problem into a small-sample image classification problem processing, and suggest the SCLMF framework based on the learner-meta-learner foundation to realize the small-sample Ethernet smart contract vulnerability detection.

In this paper, we conduct a complementary research work which consists of a joint research work on the SCSVM method and the SCLMF method. Analyzed from the perspective of the selected research problem, the real-world problems that we address in this research work are both key challenges in the current field of vulnerability detection for Ethereum smart contracts. This shows that our selected problem is of great practical significance. Analyzed from the perspective of the technical route, our proposed SCSVM method and SCLMF method are validated experimentally on publicly available smart contract datasets and compared with other typical methods. In other words, the experiments conducted validate that our complementary research can alleviate the key challenges in the area of vulnerability detection for Ethernet smart contracts. The core idea of the SCSVM method is to transform the Ethernet smart contract vulnerability detection problem into a textual classification problem, based on support vector machine technology, to achieve the detection of Ethernet smart contract vulnerabilities. The core idea of the SCLMF method transforms the Ethernet smart contract vulnerability detection problem into an image classification problem and proposes a meta-learning framework to achieve the detection of Ethernet smart contract vulnerabilities under small sample conditions. This is the first attempt to study the detection of vulnerabilities in Ethernet smart contracts under small-sample conditions, which is of great reference significance for future work on the same or similar topics.

III. GENERAL FRAMEWORK

A. SCSVM FRAMEWORK

1) THE OVERALL MODEL ARCHITECTURE OF SCSVM

The SCSVM's general model architecture is shown in Fig. 1. The model mainly consists of the following steps:

1.The regularization approach is used to the source code of the Ethernet smart contract, removing any unnecessary data like as comments.

2.The pre-processed smart contract is transformed into word vector form by word embedding technique as the model input.

3.In order to get results for vulnerability detection, an SVM model is built for extracting the features of smart contracts, the loss is calculated using the cross entropy loss function, and vulnerability classification is accomplished using a decision function.

The smart contracts are categorized using a support vector machine (SVM), and the model's objective is to identify weaknesses in Ethernet smart contracts. The smart contract

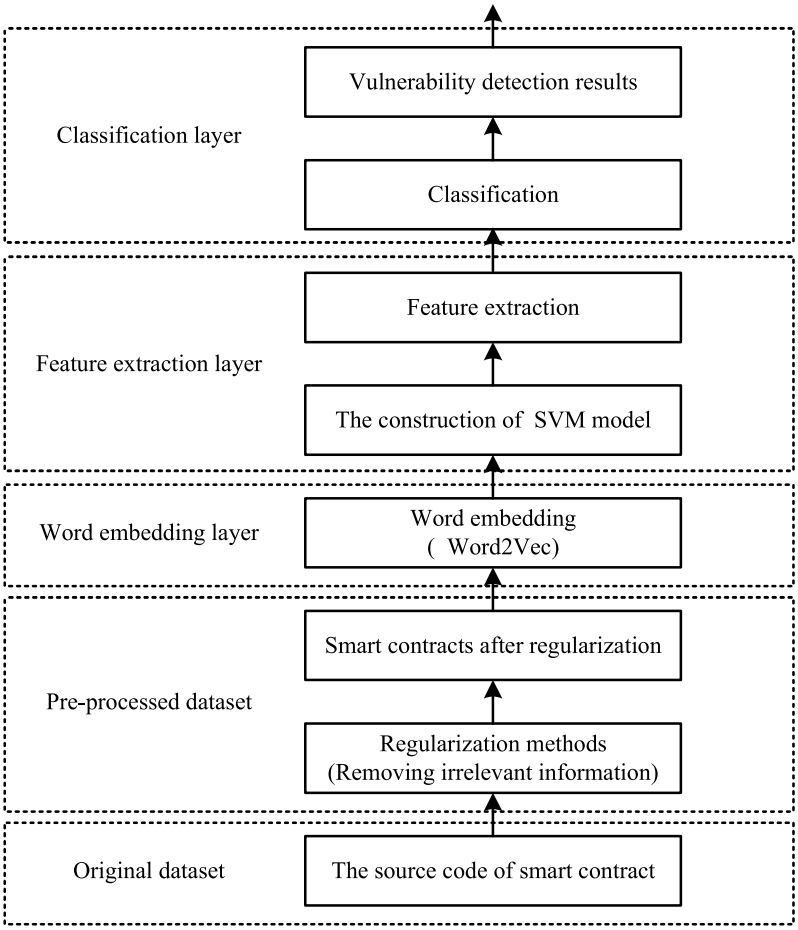


FIGURE 1. The overall model architecture of SCSVM.

source code is preprocessed in this model using the word embedding technique before being transformed into word vector form as an input to the SVM model. After computing the loss with a cross-entropy loss function, the vulnerability detection findings are derived by applying a decision function to categorize the vulnerabilities.

2) WORD EMBEDDING LAYER

The detection model proposed in this paper uses the Continuous Bag of Words (CBOW) model in the Word2Vec model for word pre-training. The Word2Vec model is used to convert smart contract text data into word vectors. the CBOW model predicts the current value by contextual information, and its network model is shown in Fig.2. This CBOW network model has window=3, $w(t)$ is the central word of the input layer, and $w(t-3)$, $w(t-2)$, $w(t-1)$, $w(t+1)$, $w(t+2)$, $w(t+3)$ are the contexts of $w(t)$. The following briefly describes the various layers of the CBOW network model:

Input layer: The context word's one-hot encoding is used as input. Assume that the context word window has a size of C and that the word vector space has a dimension of. So C*V is the input size.

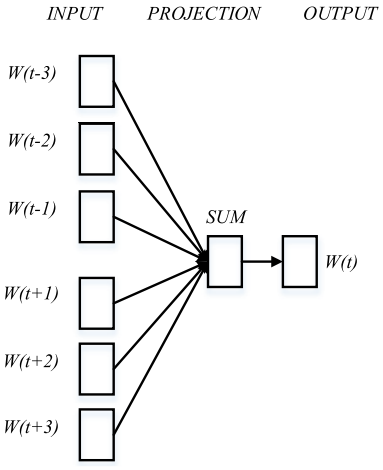


FIGURE 2. Model architecture of CBOW.

Hidden layer: Suppose the final output of the hidden layer is a word vector with dimension size N. The weight matrix between the input layer and the hidden layer is, and V*N denotes the size of W. Multiply each vector in vector group

$\{W^{d1}, W^{s2}, \ldots, W^{sC}\} \in {}^{1\times V}$ with $W \in {}^{V\times N}$ to get vector group $\{W^{d1}, W^{d2}, \ldots, W^{dC}\} \in {}^{1\times N}$ to get vector $W^d \in {}^{1\times N}$.

Output layer: initialize the output weight matrix $W^o \in {}^{N\times V}$.

The vectors $W^d \in {}^{1\times N}$ and $W^o \in {}^{N\times V}$ of the hidden layer are multiplied and processed with Softmax to output vector $W^r \in {}^{1\times V}$ Each dimension in vector $W^r \in {}^{1\times V}$ represents a word in the lexicon. The prediction target corresponds to the word represented by the index with the highest probability.

3) SVM NETWORK LAYER

Support Vector Machine (SVM) is a commonly used machine learning algorithm, mainly for classification and regression problems. Among neural networks, the SVM network layer is a special variety of network layers used for classification tasks. The input of the SVM network layer is a vector and the output is the probability that the vector belongs to different categories. The SVM network layer works by mapping the input vector into a high-dimensional space and then finding a hyperplane in the high-dimensional space to separate the vectors of different categories.The SVM network layer works by mapping the input vector into a high-dimensional space and then finding a hyperplane in the high-dimensional space to separate the vectors of different categories. The advantage of the SVM network layer is that it can handle high-dimensional data and nonlinear data and has better generalization ability.

In this experiment, we implemented the process of text classification for Ethernet smart contracts using support vector machine (SVM) and word vector techniques through code. The precise implementation of each stage in this process, which includes activities like data preprocessing, model training, and evaluation, is detailed below. First, the training and test sets' text data are read from the specified file. Next, word vector features are created for the training set's text data using the Word2Vec algorithm. Finally, the word vectors for each sample are averaged to create the feature vector representation of the sample and are saved in the DataFrame object of pandas. The word vector features and label data are read straight from the pre-processed word vector file if one already exists. Next, the feature vectors and label information are extracted from the training and test sets, and the feature vectors are processed using normalization and normalization methods to make the feature distribution of the data more normally distributed with little scale difference. The SVM model's hyperparameters, or the values of the C and gamma parameters, are then optimized using a genetic algorithm to boost the model's performance by identifying the ideal combination of hyperparameters. The accuracy of the model acquired by cross-validation is known as the fitness function, and the GA library iteratively tests various combinations of hyperparameters in accordance with its own internal rules. Finally, the output is the best result. Next, the SVM model is retrained with the best hyperparameters, predictions are made using the test set data, and evaluation metrics such as accuracy and classification reports are calculated for the model on the test set. Finally, the best hyperparameter combination, model accuracy, and classification report are outputs.

B. SCLMF FRAMEWORK

1) THE OVERALL MODEL ARCHITECTURE OF SCLMF

The overall SCLMF model architecture is shown in Fig. 3. The model mainly consists of the following steps:

1. The source code of the Ethernet smart contract is compiled by the solc tool to get the hexadecimal bytecode.

2. Get the corresponding RGB image from the hexadecimal bytecode in Python programming language.

3. Create the fundamental structure for learner-meta-learners to implement few-shot identification of vulnerabilities in ether smart contracts. The basic learner is a convolutional neural network, and the meta-learner is the MAML algorithm.

The importance of conducting static analysis work for Ethereum smart contracts [39], [40], [41]. On the one hand, for the security and vulnerability detection of smart contracts, carrying out static analysis of smart contracts can identify potential vulnerabilities and security weaknesses in the contracts and identify typical programming errors and vulnerabilities in the code. Through static analysis, problems in the contract can be detected and fixed in advance before the contract is deployed on the blockchain, thus avoiding or reducing the occurrence of security incidents. On the other hand, for the reliability and quality assurance of smart contracts, conducting static analysis of smart contracts can detect errors, anomalies, and inefficient operations in the contracts and improve the reliability and quality of the contracts. Static analysis can help developers identify problems such as dead code, uninitialized variables, and unsafe type conversions in the code and provide recommendations for fixing them, thus improving the maintainability and robustness of the contract. Therefore, it is also valuable and meaningful to use ''solc'' in our research work.

The purpose of this model is to detect vulnerabilities in Ethernet smart contracts using deep learning techniques including convolutional neural network and meta-learner MAML. it can handle small sample data and shows good performance on this task.

2) THE BASE LEARNER

The convolutional neural network CNN, which has four convolutional layers and one fully connected layer, is the key tool used in this project to define a base learner. The variable of type ''nn.ParameterList'' called ''self.vars'' contains all the tensors that need to be optimized and is of the name ''self.vars''. The weights and biases of the fully connected and convolutional layers in this model are kept in ''self.vars''. ''nn.ParameterList'', which contains the running means and variances of all BatchNorm layers that do not need to be optimized, so their ''requires_grad'' property is set to ''False''. ''nn.Parameter()'' function to create the tensor to

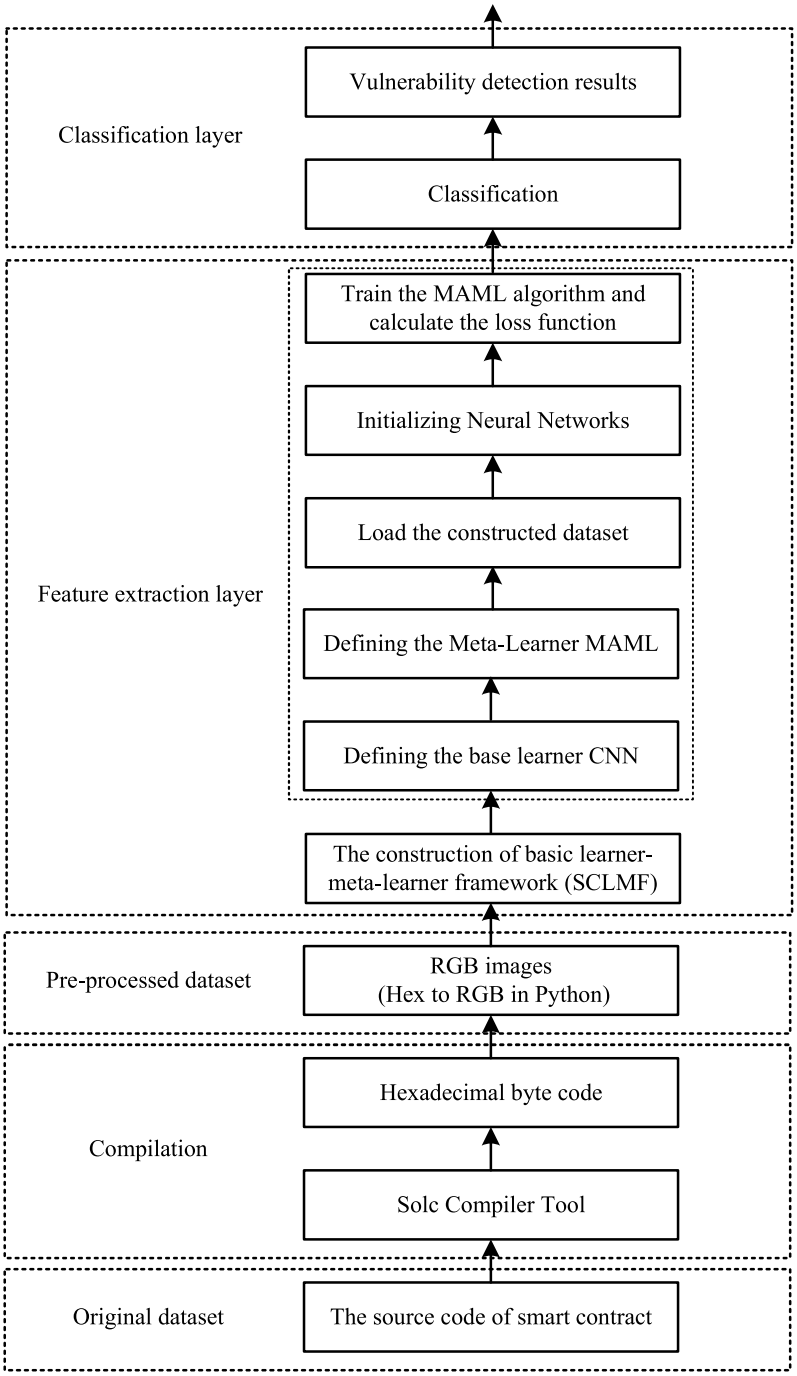


FIGURE 3. Overall model architecture of SCLMF.

be optimized, and "nn.init.kaiming_normal_()" function to initialize the weights. We also use the "nn.BatchNorm2d()" function to create the BatchNorm layers and store their running means and variances in "self.vars_bn". The "nn.BatchNorm2d()" function is used here instead of calculating the BatchNorm layers manually because PyTorch provides this function, which already implements the standard BatchNorm calculation method. The final layer we define has an input size of 64 and an output size of 5. Instead of employing the BatchNorm layer in this fully-connected layer, we just utilize a linear transformation. Fig. 4 depicts the precise model parameter structure.

3) META-LEARNER

Algorithm 1 Meta Learner describes the process of the meta-learning algorithm Meta Learner. First, in each task, the prediction result "y_hat" for the support set images is calculated by forward propagation, and the cross-entropy loss

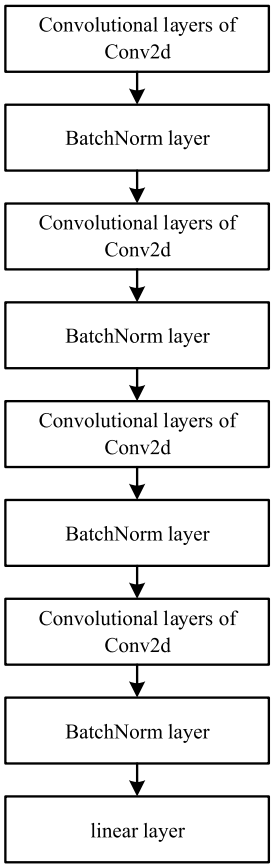


FIGURE 4. Model parameter structure of the base learner.

''loss'' is calculated. Then, the gradient ''grad'' is calculated based on the loss. Next, the fast weights ''fast_weights'' are updated by using the gradient descent method, where ''theta'' is the initial weight and ''alpha'' is the learning rate. Then, forward propagation is performed again to obtain the prediction result ''y_hat_before'' for the query set images, and the cross-entropy loss ''loss_qry_before'' and the accuracy ''correct_before'' are calculated. In the next inner loop, ''num_inner_updates'' are performed. In each inner update, the prediction result ''y_hat_inner'' for the support set images is calculated by forward propagation, and the cross-entropy loss ''loss_inner'' and gradient ''grad_inner'' are calculated. Then, the fast weights ''fast_weights'' are updated using the gradient descent method, where ''base_lr'' is the learning rate of the inner update. After completing the inner loop, forward propagation is performed again to get the prediction result ''y_hat_after'' for the query set images, and the cross-entropy loss ''loss_qry_after'' is calculated. In each inner loop, the cumulative loss ''loss_qry_after'' and the accuracy ''correct_after'' are accumulated, and then the average loss ''loss_qry_sum'' and the average accuracy ''accuracy_sum''. After all task cycles are completed, the average task loss ''loss_qry_final'' is calculated, and then the meta-model ''meta_model'' is updated by back-propagation. Finally, the loss and accuracy of ''num_inner_updates'' sub-internal cycles are calculated.

Algorithm 1 Meta Learner

Input:
Support set images, x_spt;
Support set label, y_spt;
Query set images, x_qry;
Query set label,y_qry;
Update steps,k;
Number of tasks,task_num;
meta-optimizer,meta_optim;
Initial learning rate of internal update step,base_lr;

Output:
''accuracy'' used for both internal and external loops,acc;
''query loss'' during internal loops,loss;

```
for (i=1 to task_num) do
    y_hat = forward_prop(x_spt, base_model);
    loss = cross_loss(y_hat, y_spt);
    grad = compute_gradient(loss);
    fast_weights = theta - alpha * grad;
    y_hat_before = forward_pro(x_qry, base_model);
    loss_qry_before = cross_loss(y_hat_before, y_qry);
    correct_before = compute_acc(y_hat_before, y_qry);

    for (i=1 to k) do
        y_hat_inner = forward_pro(x_spt, fast_weights);
        loss_inner = cross_loss(y_hat_inner, y_spt);
        grad_inner = compute_gradient(loss_inner)
        fast_weights = fast_weights - base_lr * grad_inner;

        y_hat_after = forward_pro(x_qry, fast_weights);
        loss_qry_after = cross_loss(y_hat_after, y_qry);
        accumulate(loss_qry_after, correct_after);
        loss_qry_after = cross_loss(y_hat_after, y_qry);
    loss_qry_sum = sum_loss_qry_after / k;
    accuracy_sum = sum_correct_after / k;
    accumulate(loss_qry_sum, loss_list_qry);
for (i=1 to task_num) do
    loss_qry_final = loss_qry_sum / num_tasks;
    backward_pro(loss_qry_final, meta_model);
    calculate_acc(num_inner_updates, loss_list_qry);
    calculate_loss(num_inner_updates, loss_list_qry);
return acc,loss;
```

4) MODEL FINE-TUNING

Algorithm 2 Finetuning describes that during model fine-tuning. First, we need to get the dimension of the input data x_spt, which can be done by calling the ''get_input_dim(x_spt)'' function. Then, we need to calculate the size of the query data, which can be done by calling the ''calculate_query_size()'' function. Next, we create an empty ''correct_list'' that stores the correct rate after each update. Then, we copy the base network ''net'' and create a new network ''new_net''. Next, we perform forward and backward propagation to calculate the loss and gradient of the new network ''new_net'' on the training data ''x_spt'', which can be done by calling ''forward_

Algorithm 2 Finetuning

Input:
Support set images, x_spt;
Support set label, y_spt;
Query set pictures, x_qry;
Query set label,y_qry;
Update steps,k;
Basic Network Model,net;
New Network Model, new_net;
Initial learning rate of internal update step,bs_lr;

Output:
List of correct rates used for both internal and external loops,accs;

```
input_dim = get_input_dim(x_spt);
query_size = calculate_query_size();
correct_list = [];
new_net = copy_network(net);
loss, grad = forward_backward_prop(new_net, x_spt);
fast_weights = update_fast_weights(theta, alpha, grad);
correct_before = forward_prop(net, x_query);
correct_after = forward_prop(new_net, x_query);
correct_list.append((correct_before, correct_after));
for (k=1 to num_inner_updates) do
    loss, grad = forward_back_prop(fast_weights, x_spt);
    f_w = update_fast_weights(fast_weights, bs_lr, grad);
    correct_after = forward_prop(f_w, x_query);
    correct_list.append(correct_after);
delete_network(new_net);
accs = [correct / query_size for correct in correct_list];
return accs
```

backward_propagation(new_net, x_spt)'' function to achieve this. Then, we use the gradient and the learning rate parameter ''alpha'' to update the fast weights ''fast_weights'', which can be done by calling ''update_fast_weights(theta, alpha, grad)'' function to achieve this. Next, we calculate the correct_before rate of the underlying network ''net'' on the query data ''x_query''. Then, we enter a loop that performs an internal update ''k'' times. In each loop, we again perform forward and backward propagation to compute the loss and gradient of the fast weights ''fast_weights'' on the training data ''x_spt''. Then, the fast weights ''f_w'' are updated using the base learning rate ''bs_lr'' and the gradient. Then, we calculate the correct rate ''correct_after'' of the fast weight ''f_w'' on the query data ''x_query'', and add it to the ''correct_list'' to the list of correct rates. At the end of the loop, we delete the new network ''new_net''. Finally, we calculate the percentage of each correct rate value in the ''correct_list'' as a percentage of the query data size ''query_size'' and store the result in the list ''accs''. This algorithm's goal is to enhance network performance for better training and testing results.

IV. EXPERIMENTS AND RESULTS ANALYSIS

A. SCSVM EXPERIMENTS AND RESULT ANALYSIS

1) EXPERIMENTAL ENVIRONMENT AND DATASET

The hardware configuration for SCSVM experiments is an Intel i7-10875H processor under Windows 11 operating system, running memory of 8 GB, and a graphics card of RTX3070. The Pytorch deep learning library is version 1.10.2, and the Python programming language is version 3.8. To further evaluate the generalization performance of the model and avoid overfitting or underfitting phenomena, the fitness function of the SCSVM approach includes cross-validation techniques in addition to the random loss function (Hinge) during the training data.

The Smartbugs [6] and Smartbugs-wild [6] public datasets are used to implement the SCSVM approach. One of them, Smartbugs-wild, has 47,587 genuine and exclusive.sol files and is created in the solidity programming language. Based on the [20] division approach, the Ethereum smart contract data is split into two categories: susceptible data and non-vulnerable data. There are two basic causes for vulnerable data: (1) Incorrect variable operations, such as integer overflow. (2) Inappropriate usage of API function calls, including timestamp dependencies and reentrance.

In the study of the SCSVM approach, a typical public dataset of smart contracts is used for experimental validation in the paper. The dataset contains typical contract vulnerabilities, such as ''Reentrancy'', ''Unchecked LL Calls'', ''Unchecked LL Calls'', ''Tx.origin'', etc. After experimental verification, the SCSVM method demonstrates better detection results compared to other smart contract vulnerability detection methods covered in the paper. Therefore, our proposed SCSVM method is effective for typical Ethereum smart contract vulnerability detection methods. Our current work on validating the effectiveness of the SCSVM method takes typical, prone, and more harmful types of vulnerable contracts as the target of our research. We believe that such research is valuable and significant.

2) PERFORMANCE COMPARISON AND RESULT ANALYSIS

To demonstrate the efficacy of the proposed SCSVM method, we compare the performance with popular static analysis tools Mythrill [20], Osiris [21], Oyente [22], Security [23], and Slither [24] and evaluate the effectiveness of the SCSVM method using accuracy, precision, recall, and F1-score typical metrics. It is shown that the SCSVM method suggested in this paper is effective. The average performance of five well-known static analysis tools on the smartbugwild dataset for Mythrill, Osiris, Oyente, Security, and Slither is displayed in Table 1 [25]. In addition, the values in Table 1 are in the form of percentages. To further compare typical tools with our proposed SCSVM approach, we computed the average performance against four vulnerability types, Reentrancy, Unchecked LL Calls, Tx.origin, and Timestamp Dependency. The reason we take this approach for comparison is that the SCSVM method is used to do smart contract

TABLE 1. Average performance of 5 well-known static analysis tools on the smartbugwild dataset.

detection method	Reentrancy				Unchecked LL Calls				Tx.origin				Timestamp Dependency			
	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
Mythrill[20]	69.02	79.14	49.69	61.05	71.45	99.85	41.36	58.49	73.07	97.46	46.94	63.37	N/A	N/A	N/A	N/A
Osiris[21]	75.42	100	49.69	66.39	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	56.72	97.54	9.62	17.52
Oyente[22]	95.65	100	91.10	95.34	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	53.54	88.13	3.14	6.07
Security[23]	82.71	94.17	68.87	79.56	84.16	100	67.43	80.54	82.49	99.62	64.95	78.64	N/A	N/A	N/A	N/A
Slither[24]	93.51	93.20	93.54	93.37	77.80	87.14	63.77	73.65	94.73	95.75	93.52	94.62	87.16	97.18	75.30	84.82

vulnerability detection by dividing two types of datasets, which are the dataset with vulnerabilities and the dataset without vulnerabilities, where the dataset with vulnerabilities contains improper use of API function calls, i.e., it contains vulnerabilities such as Reentrancy. We contrast the typical performance with the SCSVM technique we have suggested. Table 2 compares the average performance of identifying a particular vulnerability.

The SCSVM approach suggested in this paper performs well in the discovery of vulnerabilities in Ethernet smart contracts, according to a comparison of the average performance of vulnerability detection presented in Table 2. The SCSVM method's detection accuracy, recall, and F1-Score are specifically 87.55 percent, 87.68 percent, 87.46 percent, and 87.51 percent, respectively. The F1-score, a statistic that combines precision and recall and shows the overall performance of the model, is the largest for the SCSVM approach when compared to the five other methods, Mythrill, Osiris, Oyente, Security, and Slither. In the classification problem, the recall rate is used to gauge the percentage of samples with positive predictions that actually come true, while the precision rate gauges the percentage of samples with positive predictions that actually do.Since F1-score takes both precision rate and recall rate into account, it can be used to solve problems involving multiclass classification as well as unbalanced data sets. The experiments show that the SCSVM method performs brilliantly in terms of recall, with a recall rate of 87.46%, which is higher than all the remaining vulnerability detection tools; in terms of accuracy, it is only slightly worse than Slither but better than the other detection methods. These results show how the suggested SCSVM method may successfully improve the precision and effectiveness of vulnerability identification while reducing the security concerns connected with smart contracts. Besides, we also compare the SCSVM method with other deep learning methods, such as DeeSCVHunter [36], DA-GCN [37], and DR-GCN [38], to further thesis the effectiveness of the SCSVM method. The SCSVM method also shows more excellent detection results compared to three deep learning methods such as DeeSCVHunter, DA-GCN, and DR-GCN in terms of four aspects: accuracy, precision, recall, and F1-score analysis. This shows that the SCSVM method is positive and effective.

From the perspective of data set selection, most of the data we selected are from real contracts deployed on the blockchain, which can reflect the vulnerability of smart contracts in the real world. From the perspective of metrics selection, we selected accuracy, precision, completeness, and F1-score, which are reasonable and effective. From the perspective of the comparative experimental setup, we compare the SCSVM method with existing static analysis methods and typical deep learning methods, and such a setup is reasonable and effective. From the perspective of experimental results, the SCSVM method demonstrates better detection results compared with existing static analysis methods and typical deep learning methods, which confirms the effectiveness and value of the SCSVM method.

B. SCLMF EXPERIMENTS AND RESULTS ANALYSIS

1) EXPERIMENTAL ENVIRONMENT AND DATASET

The hardware setup for the SCLMF experiment in question includes an Intel i7-10875H CPU running Windows 11, 8 GB of running memory, an RTX3070 graphics card, and GPU-accelerated training models. The Pytorch deep learning library is version 1.10, and the Python programming language is version 3.8.2. The cross entropy loss function is used as the loss function in the proposed SVM-based Ethernet smart contract vulnerability detection technique SCSVM during the training data.

The dataset used for the implementation of the SCLMF method is based on the publicly available datasets Smartbugs-wild [6] and ScrawlD [7] and the Python programming language to propose the Ethernet smart contract image dataset WScrawlD, which can be used as a small sample Ethernet smart contract vulnerability detection experimental study. The Ethernet smart contract image dataset WScrawlD contains six vulnerability types, namely ARTHM, LE, RENT, TimeM, TimeO, and UE. ARTHM, RENT, TimeM, TimeO, and UE vulnerability types correspond to SWC-101, SWC-107, SWC-116, SWC-114, and SWC-104, respectively. To ensure the effectiveness of vulnerability detection, all contracts in the WScrawlD dataset are uniquely tagged, i.e., each contract corresponds to only one vulnerability type tag. In addition, small-sample image classification experiments are conducted using the publicly available dataset Omniglot [35] to compare the SCLMF method with the typical meta-learning algorithms MANN [33] and CONVOLUTIONAL SIAMESE NETS [30], as further proof of the effectiveness of the SCLMF method.

TABLE 2. Comparison of the average performance of vulnerability detection.

Average performance evaluation criteria	Tools/Methods								
	Mythrill	Osiris	Oyente	Security	Slither	**SCSVM**	DeeSCVHunter	DA-GCN	DR-GCN
Mean_Accuracy	71.18%	66.07%	74.60%	83.12%	88.3%	**87.55%**	80.50%	87.54%	81.47%
Mean_Precision	92.15%	98.77%	94.07%	97.93%	93.32%	**87.68%**	85.53%	87.15%	72.36%
Mean_Recall	45.99%	29.66%	47.12%	67.08%	81.53%	**87.46%**	74.86%	82.25%	80.89%
Mean_F1-score	60.97%	41.96%	50.71%	79.58%	86.62%	**87.51%**	79.84%	84.63%	76.39%

In our work to validate the effectiveness of the SCLMF approach, we strive to ensure that we use broad, diverse, and high-quality training data to reflect real-world smart contract vulnerabilities. The datasets we selected are from publicly available datasets and the majority of smart contract data is real and deployed on Ether. Therefore, the data we selected is real and valid and can reflect the real-world smart contract vulnerability situation.

Our proposed smart contract image dataset WScrawlD is built based on a reasonable and feasible technical approach. Our research work on the SCLMF method aims to propose a method for detecting vulnerabilities in Ethereum smart contracts under small sample conditions, and the currently constructed smart contract image dataset WScrawlD is sufficient to provide experimental data to verify the effectiveness of the SCLMF method. Because the core idea of meta-learning is to gain the ability to generalize to new tasks by learning old ones.

2) COMPARISON OF PERFORMANCE AND INTERPRETATION OF EXPERIMENTAL FINDINGS

We evaluate the effectiveness of the SCLMF method by accuracy typical metrics and compare the performance with typical meta-learning algorithms MANN [33], CONVOLUTIONAL SIAMESE NETS [30]. The dataset used for the comparison experiments with MANN, CONVOLUTIONAL SIAMESE NETS algorithm is Omniglot [35]. For the classification effect of MANN on Omniglot refer to the paper [35] and for CONVOLUTIONAL SIAMESE NETS on Omniglot refer to the paper [30].

According to our understanding, the current research progress on the detection of vulnerabilities in Ethereum smart contracts under small sample conditions is limited, and it is difficult to find a comparable comparison method. The SCLMF method is a meta-learning algorithm by nature, and comparing it with other typical meta-learning algorithms can verify the effectiveness of the method to a certain extent. Therefore, this paper conducts an empirical comparison between the SCLMF method and typical meta-learning algorithms ''MemoryAugmented Neural Networks'' and ''CONVOLUTIONAL SIAMESE NETS''. The empirical comparison of the SCLMF method with typical meta-learning algorithms ''MemoryAugmented Neural Networks'' and ''CONVOLUTIONAL SIAMESE NETS'' is conducted to further verify the effectiveness of the SCLMF method. Meanwhile, the omniglot dataset is often used as an important dataset to compare the advantages and disadvantages of meta-learning algorithms. Therefore, the omniglot dataset is chosen as the experimental dataset in this paper to further demonstrate the effectiveness of the SCLMF method.

TABLE 3. WScrawlD Few-shot classification.

SCLMF	2-way 1-shot	2-way 2-shot
Accuracy	**72.36%**	**68.16%**

Table 3 shows the small sample classification of the SCLMF method on the WScrawlD dataset. The SCLMF method performs with an accuracy of **72.36%** on the 2-way 1-shot experiment and **68.16%** on the 2-way 2-shot experiment. Due to the small number of categories and the small number of samples, only 2-category small sample experiments were performed.We examined the experimental data, and found that the main factor contributing to the 2-way-1shot's superior accuracy over 2-way-2shot was the study's primary goal, which was to enhance model performance by utilizing fewer training samples. However, the performance of the 2-way 1-shot model is superior in comparison when there are few training examples.

It can be shown that the model performs reasonably well on all test sets by looking at the 5-way 1-shot and 5-way 5-shot model training processes of the SCSVM approach using the Omniglot dataset in Fig. 5. Additionally, the SCLMF approach obtains 96.7 percent accuracy and 98.5 percent accuracy under 5-way 1-shot and 5-way 5-shot conditions, respectively, according to the experimental data, which further supports the model. This further demonstrates the model's superior performance. For the overfitting problem, SCLMF adopts a feature selection algorithm based on L1 regularization to control the model complexity and uses an early stopping strategy during the model training process to prevent overfitting. Through these measures, SCLMF can better resist the risk of overfitting.

The SCLMF method's small-sample classification results for the Omniglot dataset are displayed in Table 4. In the 5-way 1-shot trial and the 5-way 5-shot experiment, the SCLMF technique is 96.7 percent accurate. As can be shown, the SCLMF approach performs better than the MANN, CONVOLUTIONAL SIAMESE NETS algorithm for classifying tiny data.

The Omniglot dataset, in contrast, has 1623 characters (classes) from 50 other alphabets. There are 20 samples in

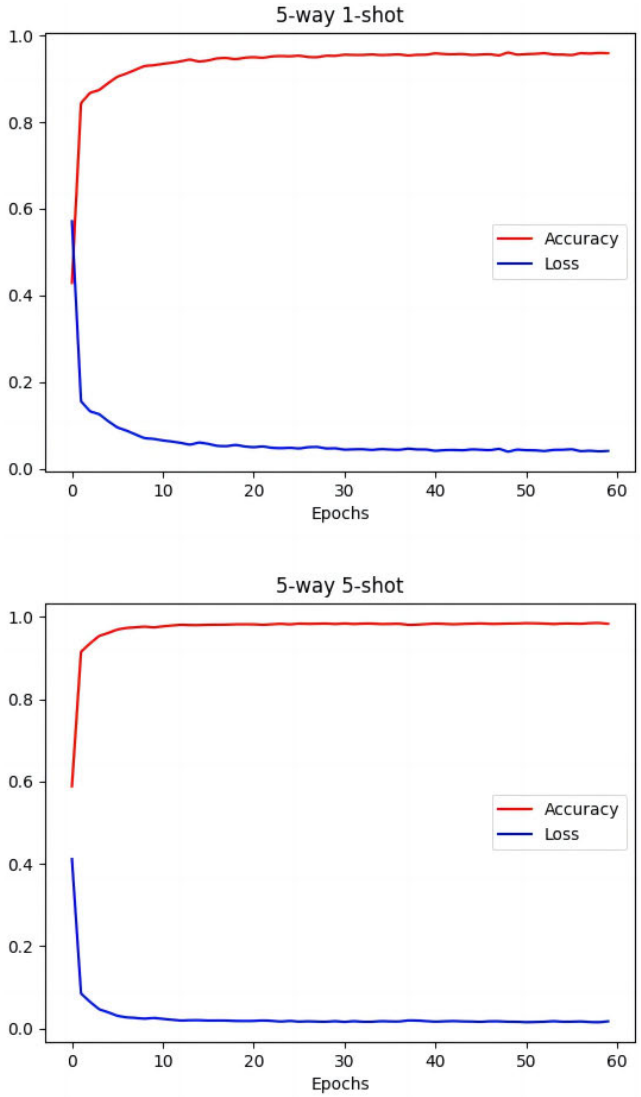


FIGURE 5. 5-way n-shot model training process.

TABLE 4. Omniglot Few-shot classification.

Detection method	5-way 1-shot	5-way 5-shot
SCLMF	**96.7%**	**98.5%**
MANN	82.8%	94.9%
CONVOLUTIONAL SIAMESE NETS	96.7%	98.4%

each class, each drawn by a different person. The WScrawlD dataset has limited categories and numbers, i.e., it contains only 6 category vulnerabilities and each category vulnerability contains only 10 images, so the classification effect has a certain gap compared with that on the Omniglot dataset due to the small number of categories of Ethernet smart contract vulnerabilities and the relatively weak learning to. To further improve the WScrawlD dataset and our method's detection performance, in future work we will examine other types of Ethereum smart contract vulnerabilities. Although there is a small gap in the detection impact of this experiment, it is confirmed that our suggested technique is efficient because the SCLMF method outperforms the MANN [33] and CONVOLUTIONAL SIAMESE NETS [30] algorithms on the Omniglot dataset.

Our proposed smart contract image dataset WScrawlD is built based on a reasonable and feasible technical approach. The smart contract data selected for the dataset WScrawlD are all from publicly available datasets and most of the smart contract data are existed and deployed on Ether. Therefore, the data we selected are real and valid and can reflect the real-world smart contract vulnerability situation. The core idea of the SCLMF method is to transform the Ethernet smart contract vulnerability detection problem into an image classification problem, so it is reasonable and effective to choose the accuracy rate as the metric. To further validate the effectiveness of the SCLMF method, we also conduct an empirical study of the SCLMF method with a typical meta-learning algorithm on the public dataset Omniglot. The main reason for comparing the SCLMF method with typical meta-learning algorithms is that the SCLMF method is also a meta-learning algorithm by nature. The results of the study demonstrate the effectiveness and value of the SCLMF method.

V. CONCLUSION

In this paper, we propose an SVM-based vulnerability detection method for Ethereum smart contracts, SCSVM, which alleviates the problem of relying on expert rules when static analysis tools detect contract vulnerabilities and the difficulty of reuse among different vulnerability types. Compared with typical static analysis methods and deep learning methods, our method has high accuracy and good classification performance with the highest F1 score. These results further emphasize the effectiveness and superiority of our proposed SCSVM approach in identifying vulnerabilities in Ethernet smart contracts. In addition, we propose SCLMF, a vulnerability detection method for Ethernet smart contracts based on the underlying learner-meta-learner framework, which alleviates the problem of relying on large-scale data when training models using deep learning methods. We combine the public dataset Smartbugs-wild with ScrawlD to build the Ethernet smart contract image dataset WScrawlD. In addition, we use WScrawlD to conduct a small-sample smart contract vulnerability detection study and achieve certain detection results. To further demonstrate the effectiveness of the SCLMF method, we conducted experiments using the method on the publicly available dataset Omniglot, which is a typical small-sample image dataset that better reflects the advantages and disadvantages of small-sample algorithms, and the experiments show that the SCLMF method exhibits excellent classification results, proving the effectiveness of the method from this perspective.

In general, the two methods proposed in this paper use technologies that belong to machine learning; both methods are centered on the study of Ethereum smart contract vulnerability detection and have achieved certain

detection results; the two methods are a complementary study on Ethereum smart contract vulnerability detection methods; the two methods target the difficult problems from different perspectives and can be adapted to smart contract vulnerability detection in different scenarios.

APPENDIX
WE OUTLINE SIX REPRESENTATIVE EXAMPLES OF SMART CONTRACT VULNERABILITIES, SUCH AS ARTHM, LE, RENT, TimeM, TimeO, AND UE

A. RENT

List 1 shows the RENT vulnerability smart contract. It is one of the more frequent and deeper threat types of smart contract vulnerabilities. Smart contracts make calls to other contracts during execution via function calls or transfers of Ether. However, there is a risk that these external calls could be exploited by a malicious attacker, causing the contract to be forced to execute the rest of the code, a process that can be thought of as reentry. The vulnerability typically occurs when a transfer function is used in the course of a smart contract. The vulnerability can lead to the theft or denial of service of tokens from the attacked contract account. The ReentranceAttack contract is shown with a simple set of code samples to analyze the reentry vulnerability. the main function of the code examples shown in the Reentrance contract is similar to the public wallet function, deposit() implements the deposit function, withdraw() implements the withdrawal function, and balanceof() implements the balance inquiry function of the Reentrance contract. In the code example shown in the ReentranceAttack contract, deposit() deposits Ether into the Reentrance of the attacked contract (step 1), withdraw() is called through attack() to realize the withdrawal operation (step 2), the code shown in the Reentrance contract makes a request() conditional judgment (step 3), the Reentrance contract calls call.value() to take the money, triggering the fallback function in the code shown in the ReentranceAttack contract (step 4), and the fallback function in the code shown in the ReentranceAttack contract is called and continues to call the ReentranceAttack contract withdraw() in the code shown to continue the withdrawal (steps 5 6), and repeat steps 3 and 4 until step 3 is not satisfied.

```
pragma solidity ^0.4.19;
contract Reentrance{
  address _owner;
  mapping (address => uint256) balances;
  function Reentrance() {
    _owner = msg.sender;
  }
  function deposit() public payable {
    balances[msg.sender] += msg.value;
  }
  function withdraw(uint256 amount)
      public payable {
    //3
    require(balances [msg.sender] >=
        amount);
    require(this.balance >= amount);
    msg.sender.call.value(amount) ();//4
    balances [msg.sender] -= amount;
  }
  function balanceof(address addr)
      constant returns(uint256){
    return balances [addr];
  }
  function wallet() constant returns(
      uint256 result) {
    return this.balance;
  }
}

pragma solidity ^0.4.19;
import "./Reentrance.sol";
contract ReentranceAttack{
  Reentrance re;
  function ReentranceAttack(address
      _target) public payable{
    re = Reentrance(_target);
  }
  function wallet() constant returns(
      uint256 result) {
    return this.balance;
  }
  function deposit() public payable{
    re.deposit.value(msg.value)();
  }//1
  function attack() public {
    re.withdraw( 1 ether);
  } //2
  //5
  function() public payable {
    if(address(re).balance >= 1 ether){
      re.withdraw(1 ether);
    }//6
  }
}
```

LISTING 1. RENT vulnerability.

B. ARTHM

Listing 2 shows the ARTHM vulnerable smart contracts, which refer to arithmetic error vulnerable contracts, in this case, integer overflow vulnerabilities, i.e., integer overflow and integer underflow. The integer overflow vulnerability corresponds to the smart contract vulnerability library SWC-101 and CWE-682, which indicates ''incorrect computation'', the vulnerability contains both integer overflow and integer underflow, integer overflow refers to the storage of values greater than the maximum support value, integer underflow refers to the storage of values less than the minimum support value. The integer overflow vulnerability is caused by the failure to logically validate the computation results in advance before the smart contract is developed. The vulnerability can have extremely serious consequences, such as infinite token increment, which means that a malicious attacker can use the integer overflow vulnerability to initiate a transaction and send a large number of tokens to a specified address with a small number of tokens. Tokens represent digital assets in the blockchain. The maintenance and upgrade of the blockchain require the participation of miners, and tokens can be paid to miners as fees. The integer overflow vulnerability is analyzed through a simple code sample. the

```
pragma solidity >=0.4.19;
contract OverFlowUnderFlow {
  uint8 public a = 0;
  uint8 public b = 2**8-1;
  function underflow() public {
    a -= 1;
  }
  function overflow() public {
    b += 1;
  }
}
```

LISTING 2. ARTHM vulnerability.

default value of a is set to 0 and the default value of b is set to 255 in the code shown in the OverFlowUnderFlow contract. The value of changes to 255 after the execution of underflow() and the value of b changes to 0 after the execution of overflow() in the OverFlowUnderFlow contract. The change of a value is the integer underflow case, and the change of b value is the integer overflow case.

C. UE

Listing 3 shows UE vulnerable smart contracts. UE vulnerable smart contracts refer to unchecked calls to vulnerable smart contracts, corresponding to the smart contract vulnerability library SWC-104, which may be vulnerable to unchecked low-level calls when using low-level functions such as send, call, callcode, and delegetecall. Because such low-level functions do not resume previous execution when the function execution fails, such an exploit could lead to an unexpected side effect where Ether is reduced but not sent out. An example of an unchecked invocation of a smart contract vulnerability is a Locking Attack. Here is a simplified example code: In the LockingAttack contract, users can deposit Ether and get their money back at any time. However, if the attacker passes a malicious contract address when calling the withdraw function, the contract will never be able to retrieve the deposit. Assuming the attacker has the malicious contract MaliciousContract when the attacker initiates a transaction to the contract LockingAttack and calls its withdraw function, it will pass the above malicious contract address and transfer some deposits at the same time. Since the withdraw function uses the call function to interact with the external contract, the attacker's malicious contract will be executed and will then throw an exception on execution, causing the funds to be locked in the contract.

```
contract LockingAttack {
  mapping (address => uint256) public
      balances;
  uint256 public totalSupply;
  function deposit() public payable {
    balances[msg.sender] += msg.value;
    totalSupply += msg.value;
  }

  function withdraw(uint256 amount)
      public {
    require(amount <= balances[msg.sender
        ], "Insufficient balance");
    require(amount <= address(this)
        .balance, "Insufficient balance
        in contract");
    // Here's the locking attack
    if (msg.sender.call.value(amount)())
        {
      balances[msg.sender] -= amount;
      totalSupply -= amount;
    }
  }
}

contract MaliciousContract {
  fallback() external payable {
    revert("Ha ha! You can't have your
        money back!");
  }
}
```

LISTING 3. UE vulnerability.

```
pragma solidity >=0.4.19;
contract Roulette {
  uint public pastBlockTime;
  constructor() public payable {}
  function () public payable {
    require(msg.value == 20 ether);
    require(now != pastBlockTime);
    pastBlockTime = now;
    if (now % 10 == 0) {
      ms.sender.transfer(this.balance);
    }
  }
}
```

LISTING 4. TimeM vulnerability.

D. TimeM

List 4 shows TimeM vulnerable smart contracts. timeM vulnerable smart contracts refer to timestamp-dependent vulnerable smart contracts, which correspond to the smart contract vulnerability library SWC-116 and CWE-829, which indicates "contains functionality from an untrusted control domain", which typically occurs when using This vulnerability typically occurs in scenarios where timestamps are used as a key element in the execution of critical events. The vulnerability arises because when a timestamp is used as a critical element in the execution of a critical event, a miner can manipulate the timestamp in his favor in a short period of time (less than 900 seconds). A simple code example analyzes the timestamp vulnerability and the code shown in the Roulette contract implements the function that if someone is the first to transfer 20 ether to the smart contract in the block whose timestamp is divisible by exactly 10, then he will get all the bets previously transferred to the contract. The block is generated by the miner, and the miner is able to know if the timestamp of the next block is divisible by 5.

E. TimeO

List 5 shows the TimeO vulnerable smart contract, which refers to the transaction order dependency vulnerability smart contract, corresponding to the smart contract vulnerability

```
pragma solidity ^0.4.19;
contract TransactionSequence{
  bytes32 constant public hash=0
      xb5b5b97fafd9855eec9b41f74dfb6c38f59
      51141f9a3ecd7f44d5479b630ee0a;
  constructor() public payable{}
  function solve(string solution) public
      {
    require(hash = = sha3(solution));
    msg.sender.transfer(100 ether);
  }
}
```

LISTING 5. TimeO vulnerability.

library SWC-114 and CWE-362, which indicates ''incorrect synchronization when using shared resources in concurrent execution''. Blockchain networks process transactions in blocks, and it takes time for transactions to propagate and for miners to agree on them. Malicious attackers use this time to monitor the transactions of the attacked contract and send their own transactions with higher gas so that their transactions are in the same block as the attacked contract transactions. The miners check the transactions within the block and give priority to the attacker contract transactions with higher gas. This results in the malicious attacker benefiting from stealing the contents of the attacked contract's transactions and the attacked contract suffering losses. The transaction order dependency vulnerability is analyzed through a simple code sample. the code shown in TransactionSequence has 100 ether in the contract account and finds the sha3 hash as 0xb5b5b97fafd9855eec9b41f74dfb6c38f5951141f9a3ecd7f44d5479b630ee0a value ''KEY'' can get 100ether. a node A in the blockchain could have used ''However, the malicious attacker monitors the ''KEY'' submitted by A through the monitoring transaction pool, and after checking its correctness, the malicious attacker uses the higher gas to send the ''After checking its correctness, the malicious attacker calls solve() with a higher gas to get the 100 ether that should belong to node A first.

F. LE

Listing 6 shows the LE vulnerability smart contract. le vulnerability refers to Ethereum locking. Here is a simple sample code for EtherLock. In this contract, we define an EtherLock contract with two parameters: beneficiary and release time. the beneficiary is the recipient of the deposit, and release time is the time that specifies when the deposit can be withdrawn. In the withdraw function, we use the required statement to check if the current time is later than the specified release time. if the condition does not hold, the function throws an exception and stops execution. Otherwise, the function transfers all the balances in the contract to the specified beneficiary address. This contract can be used to limit the use of funds or delay the release of funds. For example, suppose a project takes a certain amount of time to complete and a payment needs to be made in advance. We can use the

```
contract EtherLock {
  address payable public beneficiary;
  uint256 public releaseTime;
  constructor(address payable
      _beneficiary, uint256 _releaseTime)
      {
    beneficiary = _beneficiary;
    releaseTime = _releaseTime;
  }
  function withdraw() public {
    require(block.timestamp >=
        releaseTime, "Not yet released");
    uint256 amount = address(this)
        .balance;
    beneficiary.transfer(amount);
  }
}
```

LISTING 6. LE vulnerability.

EtherLock contract to lock this payment until the project is completed. This ensures that the money can only be used for a specific purpose and prevents accidental or misuse.

ACKNOWLEDGMENT

The results of this work are motivated by the project (No. KYZYJKKCJC23001). We would like to thank Minggang Yu, who was deeply involved in this work and provided many valuable suggestions and opinions to promote the smooth progress of this work. Thanks are also due to the other colleagues in the project team, who spared no effort in providing care and help for this work.

REFERENCES

[1] N. Szabo, ''Formalizing and securing relationships on public networks,'' *1st Monday*, vol. 2, no. 9, pp. 1–10, Sep. 1997.
[2] Q. Wang, F. Li, Z. Wang, G. Liang, and J. Xu, ''Principles and key technologies of blockchain,'' *J. Comput. Sci. Explor.*, vol. 14, no. 10, pp. 1621–1643, Oct. 2020.
[3] S. Zeng, R. Huo, T. Huang, J. Liu, S. Wang, and W. Feng, ''A survey of blockchain technology: Principles, advances, and applications,'' *J. Commun.*, vol. 41, no. 1, pp. 134–151, Jan. 2020.
[4] L. Zhang, W. Chen, W. Wang, Z. Jin, C. Zhao, Z. Cai, and H. Chen, ''CBGRU: A detection method of smart contract vulnerability based on a hybrid model,'' *Sensors*, vol. 22, no. 9, p. 3577, May 2022.
[5] L. Zhang, J. Wang, W. Wang, Z. Jin, C. Zhao, Z. Cai, and H. Chen, ''A novel smart contract vulnerability detection method based on information graph and ensemble learning,'' *Sensors*, vol. 22, no. 9, p. 3581, May 2022.
[6] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, ''Empirical review of automated analysis tools on 47,587 Ethereum smart contracts,'' in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng. (ICSE)*, Montreal, QC, Canada, Oct. 2020, pp. 530–541.
[7] C. Sujeet Yashavant, S. Kumar, and A. Karkare, ''ScrawlD: A dataset of real world Ethereum smart contracts labelled with vulnerabilities,'' 2022, *arXiv:2202.11409*.
[8] O. Levy and Y. Goldberg, ''Dependency-based word embeddings,'' in *Proc. 52nd Annu. Meeting Assoc. Comput. Linguistics*, Baltimore, MD, USA, 2014, pp. 302–308.
[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, ''Efficient estimation of word representations in vector space,'' 2013, *arXiv:1301.3781*.
[10] J. Pennington, R. Socher, and C. Manning, ''Glove: Global vectors for word representation,'' in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, Doha, Qatar, 2014, pp. 1532–1543.
[11] I. Vulić and M.-F. Moens, ''Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings,'' in *Proc. 38th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Santiago, CL, USA, Aug. 2015, pp. 363–372.

[12] G. Glavaš, M. Franco-Salvador, S. P. Ponzetto, and P. Rosso, "A resource-light method for cross-lingual semantic textual similarity," *Knowl.-Based Syst.*, vol. 143, pp. 1–9, Mar. 2018.

[13] A. B. Soliman, K. Eissa, and S. R. El-Beltagy, "AraVec: A set of Arabic word embedding models for use in Arabic NLP," *Proc. Comput. Sci.*, vol. 117, pp. 256–265, 2017.

[14] R. Laatar, C. Aloulou, and L. H. Bilguith, "Word sense disambiguation of Arabic language with word embeddings as part of the creation of a historical dictionary," in *Proc. LPKM*, 2017, pp. 1–12.

[15] C. Musto, G. Semeraro, M. De Gemmis, and P. Lops, "Learning word embeddings from Wikipedia for content-based recommender systems," in *Proc. 38th Eur. Conf. IR Res. (ECIR)*. Padua, Italy: Springer, vol. 38, Mar. 2016, pp. 729–734.

[16] A. Greenstein-Messica, L. Rokach, and M. Friedman, "Session-based recommendations using item embedding," in *Proc. 22nd Int. Conf. Intell. User Interfaces*, Limassol, Cyprus, Mar. 2017, pp. 629–633.

[17] M. Alsuhaibani, D. Bollegala, T. Maehara, and K.-I. Kawarabayashi, "Jointly learning word embeddings using a corpus and a knowledge base," *PLoS ONE*, vol. 13, no. 3, Mar. 2018, Art. no. e0193094.

[18] J. Liu, "Morpheme-enhanced spectral word embedding," in *Proc. SEKE*, Pittsburgh, PA, USA, Jul. 2017, pp. 551–556.

[19] I. Gallo, S. Nawaz, and A. Calefati, "Semantic text encoding for text classification using convolutional neural networks," in *Proc. 14th IAPR Int. Conf. Document Anal. Recognit. (ICDAR)*, Kyoto, Japan, vol. 5, Nov. 2017, pp. 16–21.

[20] B. Mueller, J. Honig, and N. Parasaram. (2017). *ConsenSys/Mythril*. Accessed: Jun. 25, 2019. [Online]. Available: https://github.com/ConsenSys/mythril

[21] C. F. Torres, J. Schütte, and R. State, "Osiris: Hunting for integer bugs in Ethereum smart contracts," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, San Juan, Puerto Rico, Dec. 2018, pp. 664–676.

[22] S. Badruddoja, R. Dantu, Y. He, K. Upadhayay, and M. Thompson, "Making smart contracts smarter," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, Vienna, Austria, May 2021, pp. 1–3.

[23] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Toronto, ON, Canada, Oct. 2018, pp. 67–82.

[24] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proc. IEEE/ACM 2nd Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, Montreal, QC, Canada, May 2019, pp. 8–15.

[25] S.-J. Hwang, S.-H. Choi, J. Shin, and Y.-H. Choi, "CodeNet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection," *IEEE Access*, vol. 10, pp. 32595–32607, 2022.

[26] H. Wu, Z. Zhang, S. Wang, Y. Lei, B. Lin, Y. Qin, H. Zhang, and X. Mao, "Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques," in *Proc. IEEE 32nd Int. Symp. Softw. Rel. Eng. (ISSRE)*, Hong Kong, Oct. 2021, pp. 378–389.

[27] M. Ren, F. Ma, Z. Yin, H. Li, Y. Fu, T. Chen, and Y. Jiang, "SCStudio: A secure and efficient integrated development environment for smart contracts," in *Proc. 30th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Tokyo, Japan, Jul. 2021, pp. 666–669.

[28] J. Huang, K. Zhou, A. Xiong, and D. Li, "Smart contract vulnerability detection model based on multi-task learning," *Sensors*, vol. 22, no. 5, p. 1829, Feb. 2022.

[29] L. Fanzhang, Y. Liu, P. Wu, F. Dong, Q. Cai, and Z. Wang, "A survey on meta-learning," *J. Comput. Res. Develop.*, vol. 44, no. 2, pp. 422–446, Feb. 2021.

[30] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *Proc. ICML Deep Learn. Workshop*, Lille, France, vol. 2, no. 1, Jul. 2015, pp. 1–30.

[31] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, Sydney, NSW, Australia, Aug. 2017, pp. 1126–1135.

[32] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Represent.*, Toulon, France, Apr. 2017, pp. 1–11.

[33] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA, Jun. 2016, pp. 1842–1850.

[34] R. Zhang, T. Che, Z. Ghahramani, and Y. Bengio, "MetaGAN: An adversarial approach to few-shot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, Montréal, QC, Canada, vol. 31, Dec. 2018, pp. 1–10.

[35] B. Lake, R. Salakhutdinov, J. Gross, J. Tenenbaum, and E. Gershman, "One shot learning of simple visual concepts," in *Proc. Annu. Meeting Cognit. Sci. Soc.*, Boston, MA, USA, vol. 33, no. 33, Jul. 2011, pp. 1–13.

[36] X. Yu, H. Zhao, B. Hou, Z. Ying, and B. Wu, "DeeSCVHunter: A deep learning-based framework for smart contract vulnerability detection," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Shenzhen, China, Jul. 2021, pp. 1–8.

[37] Y. Fan, S. Shang, and X. Ding, "Smart contract vulnerability detection based on dual attention graph convolutional network," in *Proc. Int. Conf. Collaborative Comput., Netw., Appl. Worksharing*, Virtual Event, Oct. 2021, pp. 335–351.

[38] Y. Zhuang, Z. Liu, P. Qian, T. Wang, J. Li, and X. Zhang, "Smart contract vulnerability detection using graph neural networks," in *Proc. 29th Int. Conf. Int. Joint Conf. Artif. Intell. (IJCAI)*, Montreal, QC, Canada, Aug. 2021, pp. 3283–3290.

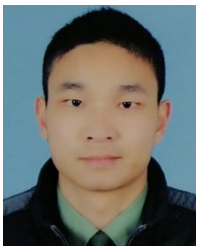
[39] Z. Liao, Z. Zheng, X. Chen, and Y. Nan, "SmartDagger: A bytecode-based static analysis approach for detecting cross-contract vulnerability," in *Proc. 31st ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2022, pp. 752–764.

[40] C. Schneidewind, I. Grishchenko, M. Scherer, and M. Maffei, "EThor: Practical and provably sound static analysis of Ethereum smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 621–640.

[41] A. Zhukov and V. Korkhov, "SmartGraph: Static analysis tool for solidity smart contracts," in *Proc. Int. Conf. Comput. Sci. Appl.*, Greece, Athens, Jul. 2023, pp. 584–598.

ZHONGJU YANG is currently pursuing the master's degree in software engineering with Army Engineer University of PLA. His research interests include requirements engineering and intelligent software testing.

WEIXING ZHU received the Ph.D. degree from the PLA University of Technology. He is currently an Associate Professor with the School of Command and Control Engineering, People's Liberation Army, Army Engineering University. His research interests include military requirements, software engineering, combat experiments, and big data.



MINGGANG YU was born in Henan, in 1986. He is an Associate Professor with the Institute of Command and Control Engineering, Army Engineering University of PLA. His research interests include SoS engineering and theory of intelligent command and control. He has published more than 50 papers in important journals and academic conferences in the research field, including 10 SCI search, 15 EI searches, and he is also the author of five books. He has chaired one National Natural Science Foundation of China, participated in the research work of several National Science and Technology Ministry Key Research and Development plan, National Key Research and Development Program of China, China Academy of Engineering Consulting Research Project, and received three awards of the Scientific and Technological Progress.