Journal of Information Security and Applications 73 (2023) 103423

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

ASSBert: Active and semi-supervised bert for smart contract vulnerability detection

Xiaobing Sun [a,b], Liangqiong Tu [a,b], Jiale Zhang [a,b,*], Jie Cai [a,b], Bin Li [a,b], Yu Wang [c]

[a] School of Information Engineering, Yangzhou University, Yangzhou, Jiangsu 225127, China
[b] Jiangsu Engineering Research Center Knowledge Management and Intelligent Service, Yangzhou, Jiangsu 225127, China
[c] Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou 510006, China

ARTICLE INFO

Keywords:
Smart contract
Vulnerability detection
Active learning
Semi-supervised learning

ABSTRACT

With the popularity of blockchain, the amount of smart contracts has increased very fast, and the safety of smart contracts has come to more extensive notice. Recently, machine learning technology has been widely applied in vulnerability detection for smart contracts. However, it implements effective smart contract vulnerability detection still faces a major challenge, that is, there is a problem of insufficient labeled data in the current field. Active learning can label data more efficiently. Nevertheless, classical active learning only uses limited labeled data for model training, contrary to the deep learning of a large amount of data required for model training. Because of the above, we provide a new framework, called *ASSBert*, that leverages active and semi-supervised bidirectional encoder representation from transformers network, which is dedicated to completing the task of smart contract vulnerability classification with a little amount of labeled code data and a large number of unlabeled code data. In our framework, active learning is responsible for selecting highly uncertain code data from unlabeled *sol* files and putting them into the training set after manual labeling. Besides, semi-supervised learning is charged to continuously pick a certain number of high-confidence unlabeled code data from unlabeled *sol* files, and put them into the training dataset behind pseudo-labeling. Intuitively, by combining active learning and semi-supervised learning, we are able to get more valuable data to increase the performance of our detection model. In our experiments, we collect our benchmark dataset included 6 vulnerabilities in about 20829 smart contracts. The result of the experiment demonstrates that our framework is superior to the baseline methods with a little amount of labeled code data and a large number of unlabeled code data.

1. Introduction

Blockchain is a new distributed system, that combines cryptographic technology, peer-to-peer (P2P) networks, and distributed deployment technology. The blockchain concept was first born in Bitcoin, and the most active blockchain system is Ethereum. Global Miners follow the consensus agreement of the blockchain to confirm transactions and record transactions in the blockchain. The bookkeeping right in the blockchain system is decentralized, and everyone has the right to bookkeep. Transaction data is chain stored in blocks in the blockchain according to the transaction execution order. Due to the data storage mode and a consensus mechanism in the blockchain network, the blockchain system is tamper-proof and decentralized. The rapid development of blockchain and its characteristics make this platform usually attacked. The core component of blockchain is smart contracts. One American's blockchain industry organization Cointegraph [1] is dedicated to analyzing blockchain ecology. Their statistics show that in March 2020, the Ethereum system deployed more than 1971000 smart contracts. Torres et al. [2] collected and conducted a comprehensive analysis of all smart contracts and transactions deployed on Ethereum between 2015 and 2020. Their analysis revealed that attacks on smart contracts occupied more than 44% of the threats to the blockchain ecosystem.

A smart contract is a protocol implemented on the blockchain and can be accessed by anyone. As long as the customary conditions are met, the protocol can be automatically executed without a trusted third party. Smart contracts programs are usually written in high-level programming languages, such as Vyper and Solidity [3]. It also inherits the anti-tampering feature of the blockchain, and cannot be changed once deployed. From the perspective of users, smart contracts are usually considered automatic guarantee accounts that can release

* Corresponding author at: School of Information Engineering, Yangzhou University, Yangzhou, Jiangsu 225127, China.
E-mail addresses: xbsun@yzu.edu.cn (X. Sun), tulq1336@163.com (L. Tu), jialezhang@yzu.edu.cn (J. Zhang), jiecaiyzu@gmail.com (J. Cai), lb@yzu.edu.cn (B. Li), yuwang@gzhu.edu.cn (Y. Wang).

https://doi.org/10.1016/j.jisa.2023.103423

Available online 13 January 2023

and transfer funds. Therefore, smart contracts often control high-value virtual currencies. If the smart contract is attacked, it will often bring huge economic losses. The famous DAO attack [4] in 2016 resulted in a loss of 60 million dollars. In 2017, Parity Wallet was attacked by two different vulnerabilities, one causing a loss of 60 million US dollars and the other freezing more than 150 million US dollars [5]. In recent years, the security of smart contracts has received extensive attention due to the endless attacks. Timely detection of smart contract vulnerabilities before deploying and invoking smart contracts is critical to the quality assurance of smart contracts.

At present, a lot of dynamic detection and static detection methods are based on many traditions of symbolic execution, program verification, static analysis and fuzzy testing [6–10]. In the dynamic detection method, dynamic symbols [11] and fuzzing test [12] are used to detect smart contract vulnerability. Existing researches are capable of effectively detecting vulnerabilities in smart contracts. Recently, the detection method of smart contract vulnerability based on machine learning is widely studied. The machine learning method greatly enhanced the efficiency of detection for the vulnerability of the smart contract. However, there are some limitations to existing researches: (1) Machine learning methods regularly require sufficient training samples, much actual data is not labeled. Moreover, manual labeling requires a large workforce and time. (2) Many researches exploit existing classification tools to label the data. This will produce large false positive rates and false negative rates. Therefore, the accuracy of the training model will be reduced. (3) In some studies, experts need to define specific defect modes or explicit rules for vulnerability detection.

To solve the problem of lacking data labeling, we proposed a framework based deep active learning method in our previous work [13], we can effectively reduce the cost of manual labeling. However, the basic idea of active learning conflicts with the model training in deep learning. Which means the labeled set in classic active learning is not enough for model training of classic deep learning. Furthermore, stream-based active learning is not suitable for the context of deep learning [14]. To cope with the problem that there are not enough labeled data from active learning for model training in deep learning. Some work is considering using a generated network to extend data or assign a pseudo label to a high-reliability sample to extend the label training set [15–17]. Some researchers combine supervised training and semi-supervised training using the labeled dataset and unlabeled data sets for active learning [18,19].

Motivated by this, we provide a new framework, called *ASSBert*, that leverages active and semi-supervised bidirectional encoder representation from transformers network. *ASSBert* can learn vulnerability detection models with strong generalization ability through collaboratively selecting the more valuable code data from the unlabeled dataset and adding them into the training dataset after manual or pseudo annotation. Specifically, *ASSBert* utilizes active learning to let experts label manually. It based on the uncertainty sampling strategy selects some *sol* files containing more information from unlabeled data sets and then merges them with the persent labeled data sets to build a training dataset. Then, *ASSBert* exploits semi-supervised learning to select some high-confidence unlabeled code data from the unlabeled dataset and combines the training set with them after pseudo labeling for further enhancement of the classification model performance. At last, *ASSBert* uses a training dataset that is continuously expanded by active and semi-supervised learning to train the deep learning model Bert as the classification model. In summary, the main contributions of this paper are as follows:

- We propose a novel active and semi-supervised bidirectional encoder representation from transformers network (*ASSBert*) framework to tackle the challenge of the scarcity of labeled code data in real-world smart contract vulnerability classification tasks.
- An active and semi-supervised learning-based Bert is developed, which works as an auxiliary term to improve smart contract vulnerability detection capability. It collaboratively selects the valuable code data from the unlabeled set and adds them to the training set after manual or pseudo annotation. Furthermore, this can reliably improve the generalization performance of classifier models.
- Empirical evaluation of the dataset we collected included 6 vulnerabilities in about 20829 smart contracts demonstrating that our framework outperforms the baseline methods with a small amount of labeled code data and available unlabeled code data.

The rest of this paper is organized as follows: Section 2 introduces some related works of vulnerability detection for smart contracts, active learning, and semi-supervised learning. In Section 3, we present the detail of our method. Section 4 describes the details of the experimental design and the results. Finally, we illustrate our discussion, conclusion, and future work in Section 5 and Section 6.

2. Related work

2.1. Vulnerability detection for smart contracts

The security of smart contracts is widely noticed, and in recent years there are many studies on the detection vulnerability of smart contracts. These methods are mainly divided into three categories: static detection method, dynamic detection method, and machine learning method.

2.1.1. Static and dynamic method

Many static and dynamic methods use the manual definitions of patterns or rules associated with the vulnerability of smart contracts and detect vulnerabilities by applying traditional methods based on static analysis, symbol execution, and fuzzy testing. In smart contract vulnerability detection, control flow analysis is the main method of static analysis. Tikhomirov et al. [6] and Feist et al. [7] have the same analysis perspective, and both use control flow graph to analyze the source code of smart contracts. Unlike the former, Vandal [20] does not analyze the source code, but analyzes the smart contract bytecode, and used static analysis to extract semantic logic relationships for security vulnerability detection. Another workflow depends on symbol execution. Luu et al. [11] firstly performed the security analysis for the smart contract using dynamic symbol execution techniques. Torres et al. [21] offered a static analysis framework focusing on integer overflow vulnerabilities. This framework first constructs a control flow graph, then performs symbolic execution, and finally performs taint analysis on the execution results to track bugs. Of cause, there are also some working based on Fuzzy tests. For example, Contractfuzzer [12] was the first working based fuzzy test. To detect security vulnerability, it defined some test oracles and generated a fuzzy input according to ABI specifications. Liu et al. [22] provided a fuzzing-based detection tool for reentrancy bugs in Ethereum. Nevertheless, these methods are low accuracy and poor flexibility because of highly dependent on fixed expert rules and patterns. These methods usually require experts to predefine corresponding vulnerability patterns in advance, which will lead to inflexible detection models and unfriendly to new vulnerability types.

2.1.2. Machine learning method

Machine learning methods usually extract the corresponding features of a smart contract and then detect vulnerabilities based on the machine learning algorithm training classification model. Most security vulnerability detection methods are supervised learning, using known vulnerability training classifiers to detect vulnerabilities. The feature extraction methods of these methods are mostly based on CFG (Control Flow Graph) or AST (Abstract Syntax Tree), focusing on the analysis of smart contract operation code, source code, or byte code. Text mining

features and smart contract security metrics are two main features used for vulnerabilities detection in smart contracts:

(1) Smart contract security metrics: In order to help smart contract developers develop new contracts, Kevin et al. [23] used the GQM (Goal Question Metric) method to analyze the metrics that can be applied to the smart contract field, and they found 15 code metrics related security. Momeni et al. [24] first analyzed the static code based on AST and CFG, proposed 17 features for model training from the perspective of code complexity, and the model can achieve 95% accuracy.

(2) Text mining features: Although most studies use different language models, the first work to be done is to extract the required features from the code data of smart contracts using different technical means. Liao et al. [25] combined the ideas of static analysis and dynamic analysis, used machine learning methods to learn code features and also developed a fuzzy module to achieve dynamic analysis. In the process of feature extraction, they did not use a single method, but used n-gram and word2vec to extract machine language features of code. Qian et al. [26] proposed an deep machine learning method. They used the BLSTM-ATT model to detect Reentrancy vulnerability. In this work, they first divided the source code into code snippets and then used the language model (word2vec) to extract code features. Similarly, Ashizawa et al. [27] also proposed a machine learning detection method based on static analysis, in which a neural network model (PV-DM) is used in the feature extraction process. Mi et al. [28] used n-gram and TFIDF technique to extract feature vectors from CFG by analyzing the bytecode of the smart contract, and then used the feature vectors to train a deep learning model to fully learn the vulnerability characteristics.

2.2. Active learning

Sampling strategy is the most important strategy of active learning [29]. The goal of active learning is to train a better model with the lowest labeling cost. The most representative candidate instances with the largest amount of information can be selected for manual labeling by reasonable application of sampling strategy. The most commonly used sampling strategy in active learning is uncertainty sampling. However, the machine learning model itself has uncertainty. In order to mitigate the negative impact of the uncertainty of machine learning models on active learning, Bayesian models are used to reduce the negative impact of model uncertainty [30–32]. The second problem to be concerned about in active learning is how to minimize human errors. Cakmak et al. [33] and Donmez et al. [34] reduced human errors by deleting error like tags, and deduced the error rate of tags by evaluating the practical ability and experience of the marker. Zhang et al. [35] asked experts to mark repeatedly to check for errors.

Currently, many research tasks apply active learning to defect prediction. Luo et al. [36] used SVM model combining clustering technology to obtain a two-stage active learning framework. Lu et al. [37] introduced an active learning method based on NaiveBayes model to reduce the limitation of supervised learning in defect detection. In order to improve the bug prediction performance among successive releases, Lu et al. [38] proposed a method of active learning as automatic model development. Zhou et al. [39] proposed a hybrid active learning method to detect cross project defects. They used mixed sampling to select informative and representative samples from unlabeled modules, and also used nonlinear mapping method to extract the feature space between two versions. All of these methods apply to defect prediction but are not applicable to the detection of vulnerabilities in smart contracts.

2.3. Semi-supervised learning

Semi supervised learning is an algorithm for model training when only a part of the samples in the dataset are labeled. The core idea of the semi supervised learning (SSL) is to utilize a large amount of unlabeled samples that are easy to obtain and to improve the learning performance of the model under the guidance of a finite labeled samples. Therefore, the model can automatically use the uncalibrated samples to improve learning performance, independent of external interactions. This has not only avoided the waste of data resources, but also solved the problem that the supervised learning method is not accurate when there is little sign sample, and that the supervised learning method is not accurate when there is no sign guidance.

The main categories of learning methods in semi supervised learning are: collaborative training, self training and expectation maximization (EM). There are a lot of studies based on semi-supervised learning, such as Arazo et al. [40] proposed a semi supervised depth convolution neural network method for hyperspectral image classification. Wang et al. [41] proposed a self training algorithm based on repeated labeling strategy, which is aimed at classifying big data in medical diagnosis, including structured, semi-structured and unstructured data. Yalniz et al. [42] studied the semi supervised learning of large convolutional networks, and used a large number of unlabeled images (up to 1 billion images) to train a high-precision image classification model. Taherkhani et al. [43] introduced a semi-supervised learning (GSSL) method based on a new type of iterative map, which training a CNN based classifier using a large number of unlabeled data and small amount of labeled data. However, they seldom pay attention to the fact that a small amount of labeled data may not enable the model to be effectively learned, and at present, semi-supervised learning is not applied to the smart contract vulnerability classification task. To solve this problem, as far as we know, we first combine semi-supervised learning and active learning into Bert deep learning algorithm to classify smart contract vulnerabilities.

Semi-supervised learning is a machine learning method that does not involve manual intervention but relies on algorithms to train models with unlabeled data. A significant disadvantage of semi-supervised learning is that the algorithm itself cannot correct its own errors, which will reduce the model generalization ability. Those instances that are incorrectly predicted will have a negative impact on the model. Active learning combined with semi-supervised learning, on the one hand, manual intervention.

3. The proposed method

In this section, we use the deep learning algorithm Bert model to carry out the smart contract vulnerability detection task. In particular, we provided a new semi-supervised active learning framework, which solves the contradiction between active learning and deep learning. First, we will provide an overview of our framework solution in section3.1. Then we will give a detailed description of the method in the following sections.

3.1. Overview

Fig. 1 shows the process of our method. It is a circular process with multiple data query rounds. In each round, the training set comes from two parts, one is an active learning query, and the other is a semi-supervised learning screening. The active learning sample query strategy will select the unlabeled sample subset for manual marking, and this process will select the first n samples with the highest uncertainty; The semi-supervised learning module attempts to select the samples that are most likely to belong to a certain category and pseudo tag them with predictive markers. This process selects the samples with the lowest uncertainty.

Here, our vulnerability detection method in the smart contract process has gone through several steps. First, give an original smart contract source code file, we need to perform some preprocessing operations on it to get the contract fragment: (a) data cleaning is necessary such as removing blank lines, and irrelevant comments; (b) redundant information that does not change the contract status needs

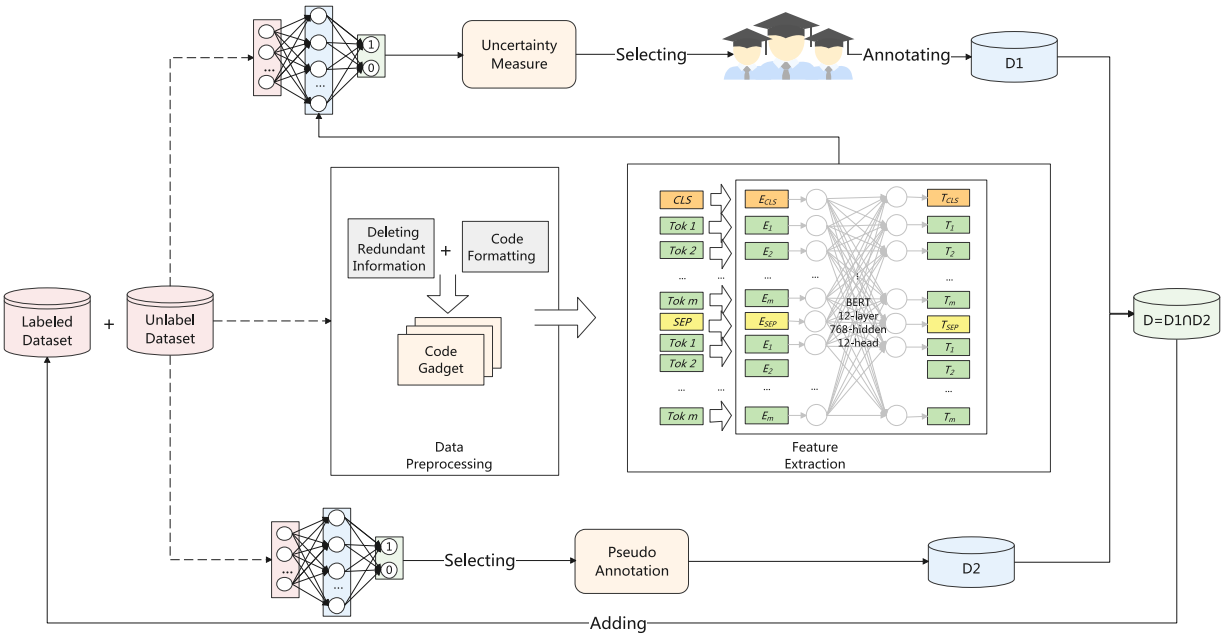


Fig. 1. Overview of the proposed smart contract vulnerability detection method.

to be deleted; (c) To make it easier to use regular, we need to format the cleaned contract code. Second, we collected this vocabulary from the official language description released by Ethereum [44] and added it to the vocabulary of the Bert model. For each sequence, a `[cls]` token is pre-added to indicate the beginning of the gadget, and a `[sep]` token is attached to indicate the end of the code line. Then, we analyze all of the smart contract code gadgets as a series of code representations inserted into the eigenvector representation. The third step is the process of iteratively extending the training set and updating the model. The active learning module uses the uncertainty sampling strategy selecting the sample with the largest amount of information from the unlabeled set, and requires experts to add their labels for the next training. The semi-supervised module selects the most confident sample subset of the model from the unlabeled set by using pseudo tags and adds it to the next round of training set. The last step is to stop iteration until the target effect is achieved.

3.2. Model training

3.2.1. Code formatting

Regular expressions can only be matched line by line when matching source code. It is difficult to accurately match code that spans more lines, so the source code format may not be suitable for using regular expressions directly. The code of a smart contract is written by developers, and the coding habits of developers are similar, resulting in a variety of formats of smart contract source code files. In order to use regular expressions more easily and match ideal fields more accurately, we need to format the source code first. Source code formatting is mainly used to normalize the non-standard writing of developers, such as the code line of the line breaking is not line breaking, and the function input parameters are written in separate lines. The main steps are as follows:

(1) Redundant spaces or empty lines in code statements will be deleted. Multiple consecutive spaces in a statement will be reduced to one space, and redundant line breaks will be deleted, leaving at most one line break.
(2) Each line of formatted code is marked with a semicolon (;), and the left brace ({) or the right brace (}) ends. The function input parameter is changed from a multi-line format to one line and ends with the left brace ({).
(3) After the above steps formatting, the (}) at the end of a piece of code should be written on another line.
(4) The smart contract code is written by developers. There is a situation where the writing is not standardized and words are made freely. It is necessary to replace the defined function name and variable name with *FUN1... FUNn*, *VAR1...VARn*.

3.2.2. Data preprocessing

The maximum input token length of the Bert model is 512. In order to minimize the code length and avoid missing important information during feature extraction. We need to preprocess the source code to retain the statements most related to the vulnerability as much as possible and delete the parts that do not change the state of the smart contract and are irrelevant to the contract execution logic. According to the development document of Ethereum's official programming language solidity, we summarize the following parts that need to be deleted from the source file, contract level, and function level.

(1) Source file level

i. Related to the keyword pragmas: the code describing the version of the program; the code appointing ABI to encode or decoder and the code for SMT checker.
ii. Related to the code for importing other source files.
iii. And need to delete comment lines in various formats.

(2) Contract level

i. The code that describes the event. The event is the interface of the Ethereum virtual machine logging function and has nothing to do with the contract execution logic.
ii. Functions with empty structure, that is, declarative functions, which need to be inherited and rewritten.
iii. Library definition, the library cannot have state variables and cannot accept Geth, and the smart contract state will not be changed.

(3) Function level

i. Statements used to trigger events.
ii. Functions of pure and view types cannot change the status of smart contracts.

3.2.3. Feature extraction

The machine cannot directly understand the semantics of the text, and cannot directly use characters for operations. Therefore, we need to digitize the text. First, we need to segment the sentences and convert the sentences into word sequences. In the Bert model, word sequences begin with token CLS and end with token Sep. In addition, some keywords with specific meanings in the language of solidity are not included in the vocabulary of the Bert model. Therefore, we need to collect this vocabulary from the official language description released by Ethereum and add it to the Bert vocabulary. The input of the Bert model is a fixed-length sentence, and the length of each code gadget is not unique. Afterword segmentation, the sequence needs to be padded. For sentences that are not long enough for the maximum length, they need to be supplemented with 0, and sentences that exceed the maximum length will be automatically truncated. After the above operations, each code gadget can be embedded into a fixed-length feature vector.

3.3. Active learning with uncertainty measure

The goal of active learning in smart contracts vulnerability detection task is to extend the labeled dataset by selecting the most information samples and create a classification model. Active learning has many strategies for selecting samples, such as query-by-committee, uncertainty sampling, error reduction, and density weighting. These policies vary greatly in selecting the most informative samples. Uncertainty sampling is the most common sampling method. In our method, expert labeling is not a real labeling action, but a simulation of an expert labeling action by querying the real label of a sample. From the viewpoint of machine learning, the uncertainty sampling strategy does select the most uncertain samples in the model from the marker set. In other words, the prediction result of the model is 0.5, and the model cannot identify which type the sample belongs to. In our approach, $P(.)$ is classification model based on active learning.

$$P(y^*|x^*, L) = \int P(y^*|x^*, \omega)P(\omega|L)d\omega \tag{1}$$

Where $L = \{x, y\}$ specifies that the training dataset. The distribution of model parameters is represented by ω.And $y = \{0, 1\}$ is the set of labels data. In our study, $y = 1$ indicates a smart contract containing one or more vulnerabilities, and $y = 0$ indicates no vulnerabilities.

Because the previous distribution $P(\omega|L)$ is too complicate to calculate directly, the function $q(\omega|\theta)$ is controlled by the parameter group $\theta = (\mu, \sigma)$, which is close to the following distribution $P(y^*|x^*, D)$. The parameter $\theta = (\mu, \sigma)$ is a normal distribution. And μ and σ are the mean and standard deviation of the distribution. KL divergence can optimize the distance between $q(.)$ and $P(\omega|L)$ functions to obtain the following results:

$$\theta^* = arg\min_{\theta} \mathbb{E}_{q(\omega|\theta)} \left[log(\left[\frac{q(\omega|\theta)}{P(L|\omega)P(\omega)} \right]) \right] \tag{2}$$

The function of minimization of KL divergence is transformed into solving the maximization function $ELBO(.)$ and the transformed function is:

$$ELBO(q) = \mathbb{E}\left[log(P(\omega))\right] + \mathbb{E}\left[log(P(L|\omega))\right] - \mathbb{E}\left[log(q(\omega|\theta))\right] \tag{3}$$

Methods for measuring uncertainty include: If you refer to the custom symbol variable y, the uncertainty measurement function $H(.)$ of the example $\vec{x}_i$ is defined as the conditional entropy.

$$H(y|x) = -\sum_{y \in Y} P(y|x_i, \theta_L) log P(y|x_i, \theta_L) \tag{4}$$

where θ_D represents that classification model θ is trained based on subset D. And the prediction result of model calculation is $P(y|x_i, \theta_D)$.

Algorithm 1 *ASSBert* For Smart Contract Vulnerability Detection

input: The labeled source code of smart contract L; The unlabeled code gadget of smart contract U; The probabilistic classifier θ
Output: Three Indicators of classifier θ

function *ASSBert*(L, U, θ)
  repeat
    D_1, U = selectAL (L, U, θ)
    D_2, U = selectSSL (L, U, θ)
    L=$D_1 + D_2$
    Train the classifier model θ based on L
  until Meeting the stop criterion
  return The result of three indicators
end function
function SELECTAL(L, U, θ)
  repeat
    for $x_i \in U$ **do**
      Select instance x_i according to Eq. (1)
      Query the label y_{x_i} of x_i
      Remove x_i from $U(U' = U - x_i)$, U=U'; Merge (x_i, y_i) into $D_1(D_1' = L + (x_i, y_i))$, D_1=D_1'
    end for
  until Meeting the stop criterion
  return D_1, U
end function
function SELECTSSL(L, U, θ)
  for $x_i \in U$ **do**
    Select x_{d2} from U according to classifier θ
    Pseudo label x_{d2}
    Remove $\{(x_{d1}, y_{d1}), (x_{d2}, y_{d2})\}$ from $L(L' = L - \{(x_{d1}, y_{d1}), (x_{d2}, y_{d2})\})$; Merge (x_r, y_r) into $D_2(L' = D_2 + (x_r, y_r))$
  end for
  return D_2, U
end function

3.4. Semi-supervised learning module

Each round of training set in the framework is divided into two parts, one part is selected by active learning strategy, and the other part is from a semi-supervised learning module. In semi-supervised learning, there is an important concept of pseudo-labeling, because it cannot be judged whether the label predicted by the model is correct. In semi-supervised learning, model prediction results are often regarded as pseudo-labeling of samples. Our framework is divided into active learning modules and semi-supervised learning module. After the active learning module stops its cycle, then the next module is the semi-supervised learning module. The initial training set T of the semi-supervised learning module is the labeled subset $D1$ obtained by the active learning module. And the semi-supervised learning module is also a cyclic iterative process until the effect of the classification model is no longer improved. In more detail, the semi-supervised learning module is divided into the following steps. Firstly, the supervised model M1 is trained using the training set T; Secondly, the trained model $M1$ is used to predict the unlabeled sample set to obtain the prediction probability of each sample; Next, we select n sample subsets $D2$ with the highest confidence from the unlabeled sample set based on the prediction probability; Thirdly, we get the pseudo label of the sample subset $D2$ based on the model prediction results, and then update the semi-supervised learning module training set $T = D1 + D2$ to train the model $M1$ again; Finally, repeat the above steps until the effect of the classification model is not improved.

Table 1
Descriptive statistics for smart contract vulnerability types grouping.

Vulnerability type	#Addresses	#Contracts	Containing types
Timestemp(Vuln1)	452	3308	Time Manipulation, Bad Randomness
CallDepth(Vuln2)	403	1998	Unchecked Low Level Calls
Reentrancy(Vuln3)	552	4312	Reentrancy
TOD(Vuln4)	937	5658	Front Running
Flow(Vuln5)	1351	4773	Arithmetic
Tx.origin(Vuln6)	149	780	Access Control
All	3844	20829	–

4. Experiment

4.1. Experiment set and benchmark dataset

Experiments were conducted using a 10 fold crossing verification. Each experiment randomly divides data sets. Avoiding duplication between test data and training data, we chose 30% data sets are used as test data, and other data is used as training data. Extract fixed percentage data for manual labeling from training data. Before this experiment, we need to take 5% of the dataset as the initial labeled data for the initial model training. Except for the initial training data set, the remaining training data in the data set is used as the unlabeled sample pool. At the stage of active learning, the process costs a query label if the process can make vulnerability predictions more effective and improve the quality of the software, but if the number of queries is controlled to a small number, it can be suppressed to less than 20% of the total number of unlabeled samples, this cost is accepted [38]. In order to control the marking cost within the acceptable range, we have selected four thresholds, namely 5%, 10%, 15% and 20%. In real practice, the samples to be labeled that selected by the model need to be labeled by domain experts. However, in our experiment, we did not really ask experts to label, but like works [39,45], we directly obtained their labels from the benchmark dataset. This is used to simulate the action of a human inspector to check the source code file.

The NCC Group [46] once ranked distributed application vulnerabilities and summarized the top 10 vulnerability types, named Decentralized Application Security Project (DASP) TOP10 [46]. We selected 6 vulnerabilities from the top 10 summarized by them, and the details are in the Table 1. In order to get enough experimental data, we collected three public datasets: Smartbugs [47], SoliAudit-benchmark [48] and SolidiFi-benchmark [49]. And we delete duplicate data and data that cannot find contract source code. Finally, we collected a total of 20829 contracts covering 6 vulnerabilities that need to be detected, and then we took the collected data as the final experimental data set. Then we use Oyente [11] and SmartCheck [6] to label our all datasets and get our baseline dataset. We use 1 or 0 to indicate whether there is a vulnerability. If a smart contract is marked as 1, it means that the contract contains at least one vulnerability. In any case, the code of the contract has no vulnerability and is a secure contract. In addition, the Table 1 lists the details of our implementation data.

4.2. Performance indicators

In smart contract vulnerability detection [24–26,28], accuracy, precision and recall are widely used in model performance evaluation. Among them, accuracy is the most commonly used measure of performance evaluation indicators. However, due to class imbalance in our dataset (a small number of vulnerable smart contracts are non vulnerable), we cannot use only one indicator to measure model performance. We decided to integrate the other two indicators in the experiment and use accuracy, accuracy and recall rate to comprehensively evaluate. The calculation method of measurement indicators is calculated by confusion matrix shown in Table 2. We will introduce the calculation formula of each indicator and its meaning in detail.

Table 2
Confusion matrix.

	Predicted as vulnerability	Predicted as non-vulnerability
Actual vulnerability	TP	FN
Actual non-vulnerability	FP	TN

(1) Accuracy Accuracy refers to the percentage of correctly predicted samples (including those with vulnerabilities and those without vulnerabilities) in the total samples. Its value range is between [0,1]. The greater the accuracy value, the better the prediction ability of the model.

$$accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)} \tag{5}$$

(2) Precision Precision refers to the proportion of positive samples correctly predicted in all correctly predicted samples. Its value range is also between [0,1]. Similarly, the greater the precision value, the better the prediction ability of the model.

$$precision = \frac{TP}{(TP + FP)} \tag{6}$$

(3) Recall Recall rate is the ability of the model to correctly predict the number of true positive samples. The calculation method is the proportion of correctly predicted true positive samples in all predicted positive samples. The larger its value is, the more real vulnerable data can be found in the model.

$$recall = \frac{TP}{(TP + FN)} \tag{7}$$

4.3. Comparison with different methods

Bert [50] is one of the most commonly used natural language processing models in vulnerability detection research. Bert is a typical bidirectional coding model, which utilizes transformer encoder blocks for connection, and can fully consider the context information of a text. Since the code is a special piece of text, it is naturally considered that the excellent natural language processing model (Bert) can be used for vulnerability detection. In order to verify that the combination of semi supervised active learning method can improve the efficiency of smart contract vulnerability detection, We will compare the capabilities of our method and the other two baseline models under different proportions of labeled datasets. These baseline methods are Bert, a deep learning algorithm with no active and semi-supervised learning, Bert with active learning but no semi-supervised learning(Bert-AL), and Bert with semi-supervised learning but no active learning(Bert-SSL).

The experimental results for smart contract vulnerability detection on our collected six benchmark datasets with our framework and baselines are shown in Table 3 in detail. We conducted experiments and recorded the results when the ratio of labeled samples was 5%,10%,15%, and 20% respectively. To reduce the error, we will repeat the experiment 10 times for each algorithm and then calculate the average value as the final result. It can be obviously noted that when less manual labeling is used, the Bert model with active learning has better performance than that without active learning. And in the case of limited labeled data sets, the method based on semi-supervised learning can slightly improve the performance of the classification model. The framework *ASSBert* we proposed can improve the performance of the classification model significantly. Especially when the labeling ratio is higher than 10%, the *ASSBert* framework is more effective. It can be clearly seen from the table that the accuracy of the model is approximately 50%. Especially for Timestamp, *ASSBert* has a much higher effect than the other three methods. Our method is 40.9%, 41.6%, and 54.5%; While Bert-SSL was 11.6%, 16.4%, 32.1%; Bert-AL was 30.9%, 41.6% and 16.1%; Bert was 31.1%, 12.9%, and 45.8%.

Table 3
The Detailed Results for *ASSBert* and other baseline methods with different labeling.

Dataset	Methods	Number of labeling											
		5%			10%			15%			20%		
		a	p	r	a	p	r	a	p	r	a	p	r
Timestamp	Bert	0.274	0.119	0.156	0.311	0.129	0.458	0.461	0.240	0.182’	0.721	0.491	0.435
	Bert-AL	0.396	0.286	0.266	0.309	0.416	0.161	0.534	0.534	0.467	0.791	0.421	0.659
	Bert-SSL	0.386	0.286	0.574	0.116	0.164	0.321	0.534	0.534	0.467	0.784	0.421	0.659
	ASSBert	0.233	0.254	0.323	0.409	0.416	0.545	0.461	0.651	0.618	0.786	0.574	0.730
CallDepth	Bert	0.274	0.119	0.443	0.233	0.173	0.134	0.300	0.500	0.179	0.721	0.491	0.435
	Bert-AL	0.286	0.396	0.266	0.300	0.500	0.179	0.534	0.534	0.467	0.891	0.421	0.659
	Bert-SSL	0.286	0.396	0.266	0.355	0.372	0.177	0.534	0.534	0.467	0.891	0.421	0.659
	ASSBert	0.333	0.254	0.223	0.433	0.546	0.560	0.551	0.651	0.618	0.890	0.517	0.730
Reentrancy	Bert	0.274	0.119	0.056	0.311	0.129	0.134	0.513	0.314	0.497	0.721	0.491	0.435
	Bert-AL	0.286	0.396	0.266	0.323	0.287	0.124	0.534	0.534	0.467	0.791	0.521	0.659
	Bert-SSL	0.286	0.396	0.266	0.430	0.172	0.123	0.534	0.534	0.467	0.871	0.577	0.559
	ASSBert	0.333	0.254	0.223	0.504	0.430	0.323	0.677	0.611	0.658	0.790	0.517	0.730
TOD	Bert	0.174	0.229	0.056	0.311	0.129	0.134	0.513	0.314	0.497	0.721	0.535	0.491
	Bert-AL	0.386	0.266	0.396	0.447	0.205	0.155	0.534	0.534	0.467	0.733	0.521	0.587
	Bert-SSL	0.286	0.396	0.266	0.423	0.102	0.121	0.534	0.534	0.467	0.717	0.559	0.469
	ASSBert	0.333	0.254	0.223	0.523	0.202	0.221	0.551	0.651	0.618	0.851	0.622	0.630
Flow	Bert	0.108	0.206	0.124	0.278	0.374	0.129	0.513	0.314	0.497	0.630	0.558	0.514
	Bert-AL	0.216	0.433	0.110	0.216	0.433	0.110	0.534	0.534	0.467	0.714	0.638	0.623
	Bert-SSL	0.286	0.396	0.266	0.505	0.122	0.124	0.534	0.534	0.467	0.705	0.544	0.631
	ASSBert	0.308	0.306	0.124	0.478	0.374	0.329	0.551	0.651	0.618	0.857	0.617	0.658
TxOrgin	Bert	0.370	0.297	0.157	0.447	0.205	0.255	0.513	0.314	0.497	0.721	0.491	0.435
	Bert-AL	0.417	0.089	0.101	0.446	0.194	0.248	0.534	0.534	0.467	0.733	0.679	0.442
	Bert-SSL	0.329	0.094	0.116	0.425	0.287	0.335	0.534	0.534	0.467	0.793	0.526	0.597
	ASSBert	0.446	0.194	0.148	0.545	0.546	0.560	0.551	0.651	0.618	0.831	0.602	0.763

When the percentage of maximum queries (labeling ratio) is set to 20%, we will analyze the following results in detailedly. When the labeling ratio is 20%, the accuracy of Bert-AL(Bert only with active learning) and Bert-SSL(Bert only with semi-supervised learning) is higher than 75%, and even worse the accuracy of Bert is lightly lower than 75%. But none of them is as accurate as the *ASSBert*, and the accuracy of our proposed methods(*ASSBert*) is basically higher than 80%. In terms of Timestamp, TOD, Flow, and Tx.orgin vulnerabilities, the accuracy of our method(*ASSBert*) is slightly higher than that of Bert-Al and Bert-SSL. However, for CallDepth and redundancy vulnerabilities, the accuracy of our framework is not as good as the latter two methods. Bert Al was 89.1% and 79.1% respectively; Bert-SSL was 89.1% and 87.1% respectively, and our method(*ASSBert*) was 89% and 79% respectively. The framework *ASSBert* combines semi-supervised learning module and active learning module to make full use of unlabeled data sets, thus effectively reducing the cost of labeling and improving the detection efficiency of the model.

4.4. Comparison with different sampling strategies

To verify that the uncertainty sampling strategy can improve the superiority of our framework more than other active learning sampling strategies, we let *ASSBert* train based on different sampling strategies and compare their results. The five different sampling strategies are: Random, QBC, Density, EER, Unc. Random is a method of violently randomly choosing a sample. The query by committee (QBC [51]) method uses multiple models instead of one model. A model set is called a committee. The most controversial data point is selected by voting as the next data point to be marked. Density-Weighted Method considers that the data that is dense and difficult to distinguish is more valuable. Expected error reduction (EER [52]) can select the sample data that will reduce the loss function the most through adding a sample. The query method of uncertainty sampling (Unc) is to extract the sample data that is difficult to distinguish from the model and provide it to experts. Tables 4–9 shows detail results of our experimental under different sampling strategies for smart contract vulnerability detection on our collected six benchmark datasets, when the ratio of labeled samples varies between 5%,10%,15%. It can be observed that uncertainty based sampling has the highest performance improvement compared with other sampling strategies, and it can be clearly seen that uncertainty based sampling strategies have better effects than other strategies. Although our framework based on the uncertainty sampling strategy is not completely better than other strategies, overall, uncertainty sampling can make the model achieve better results. Especially for vuln2 (CallDepth)and vuln4 (TOD), the proportion of labeling reaches 20%, and the accuracy of the uncertainty sampling strategy model reaches 89%, nearly 10% higher than that of random sampling and QBC. However, under the uncertainty sampling strategy, the performance of the model is not as high as that of other strategies. More detailed results can be queried in Tables 4–9.

Table 4
Performance Comparison of *ASSBert* under different strategies on Vuln1.

Measures	Labeling	Random	QBC	Density	EER	Unc
		%				
Accuracy	5	16.9	23.2	19.8	26.7	22.3
	10	33.1	36.7	37.2	29.7	40.9
	15	43.0	45.9	40.5	44.8	46.1
	20	63.4	74.0	65.1	66.7	78.6
Presion	5	28.2	37.2	23.3	19.0	25.4
	10	47.7	49.3	35.8	31.2	41.6
	15	54.7	60.9	58.9	61.7	65.1
	20	49.3	58.5	66.5	55.0	57.4
Recall	5	15.8	27.9	31.2	33.1	32.3
	10	29.7	44.3	43.3	50.8	54.5
	15	44.8	58.1	56.4	60.0	61.8
	20	60.9	70.5	74.9	69.4	73.0

4.5. Performance of ASSBert under different levels of data preprocessing

The maximum input token length of the Bert model is 512. In order to minimize the code length and avoid missing important information during feature extraction. We need to preprocess the source code to retain the statements most related to the vulnerability as much as possible and delete the parts that do not change the state of the smart contract and are irrelevant to the contract execution logic. According to

Table 5
Performance Comparison of *ASSBert* under different strategies on Vuln2.

Measures	Labeling	Random	QBC	Density	EER	Unc
			%			
Accuracy	5	21.3	28.1	35.6	25.5	33.3
	10	27.6	43.3	35.5	22.1	43.3
	15	43.3	35.8	48.4	54.7	55.1
	20	79.9	78.3	85.2	80.2	89.0
Presion	5	10.8	14.5	20.2	22.1	25.4
	10	27.8	19.7	52.1	36.2	54.6
	15	58.3	70.5	68.6	61.3	65.1
	20	66.3	50.6	51.7	46.5	51.7
Recall	5	10.9	20.7	22.2	11.5	22.3
	10	49.6	43.3	53.9	37.6	56.0
	15	60.9	54.4	61.7	60.9	61.8
	20	66.6	73.8	61.7	71.7	73.0

Table 6
Performance Comparison of *ASSBert* under different strategies on Vuln3.

Measures	Labeling	Random	QBC	Density	EER	Unc
			%			
Accuracy	5	35.5	30.0	28.1	27.6	33.3
	10	47.7	48.5	50.4	43.3	50.4
	15	65.2	64.6	67.7	57.7	66.7
	20	78.3	80.2	79.0	73.1	79.0
Presion	5	17.4	26.4	12.2	12.4	25.4
	10	37.2	35.7	43.2	42.7	43.0
	15	59.8	63.3	63.4	57.4	66.1
	20	66.4	53.8	63.4	54.7	51.7
Recall	5	13.4	11.6	11.9	20.2	22.3
	10	30.0	33.3	27.8	30.7	32.3
	15	58.6	63.8	59.3	62.6	65.8
	20	66.3	74.5	71.6	69.3	73.0

Table 7
Performance Comparison of *ASSBert* under different strategies on Vuln4.

Measures	Labeling	Random	QBC	Density	EER	Unc
			%			
Accuracy	5	21.6	23.2	37.2	21.6	33.3
	10	47.7	42.4	48.5	42.3	52.3
	15	43.0	30.9	53.4	50.4	55.1
	20	74.9	82.9	85.0	77.5	85.1
Presion	5	23.3	16.9	20.2	19.8	25.4
	10	44.3	31.2	20.0	23.3	20.2
	15	52.9	61.3	54.5	48.8	65.1
	20	63.0	58.8	61.3	63.4	62.2
Recall	5	11.0	36.7	22.1	15.4	22.3
	10	12.2	35.5	17.3	3.8	22.1
	15	54.7	56.6	60.3	50.5	61.8
	20	57.6	60.4	59.8	61.7	63.0

Table 8
Performance Comparison of *ASSBert* under different strategies on Vuln5.

Measures	Labeling	Random	QBC	Density	EER	Unc
			%			
Accuracy	5	27.2	25.2	33.4	22.6	30.8
	10	31.2	43.5	45.0	35.8	47.8
	15	51.4	50.9	56.1	52.5	55.1
	20	81.2	76.6	80.8	76.0	85.7
Presion	5	29.0	27.7	23.3	23.3	30.6
	10	36.2	35.5	24.0	31.6	37.4
	15	60.1	53.4	63.8	55.7	65.1
	20	54.4	55.8	60.4	63.4	61.7
Recall	5	9.6	4.5	12.6	16.4	12.4
	10	23.3	22.2	30.1	29.0	32.9
	15	60.6	56.6	62.6	48.1	61.8
	20	63.8	55.7	55.2	57.6	65.8

Table 9
Performance Comparison of *ASSBert* under different strategies on Vuln6.

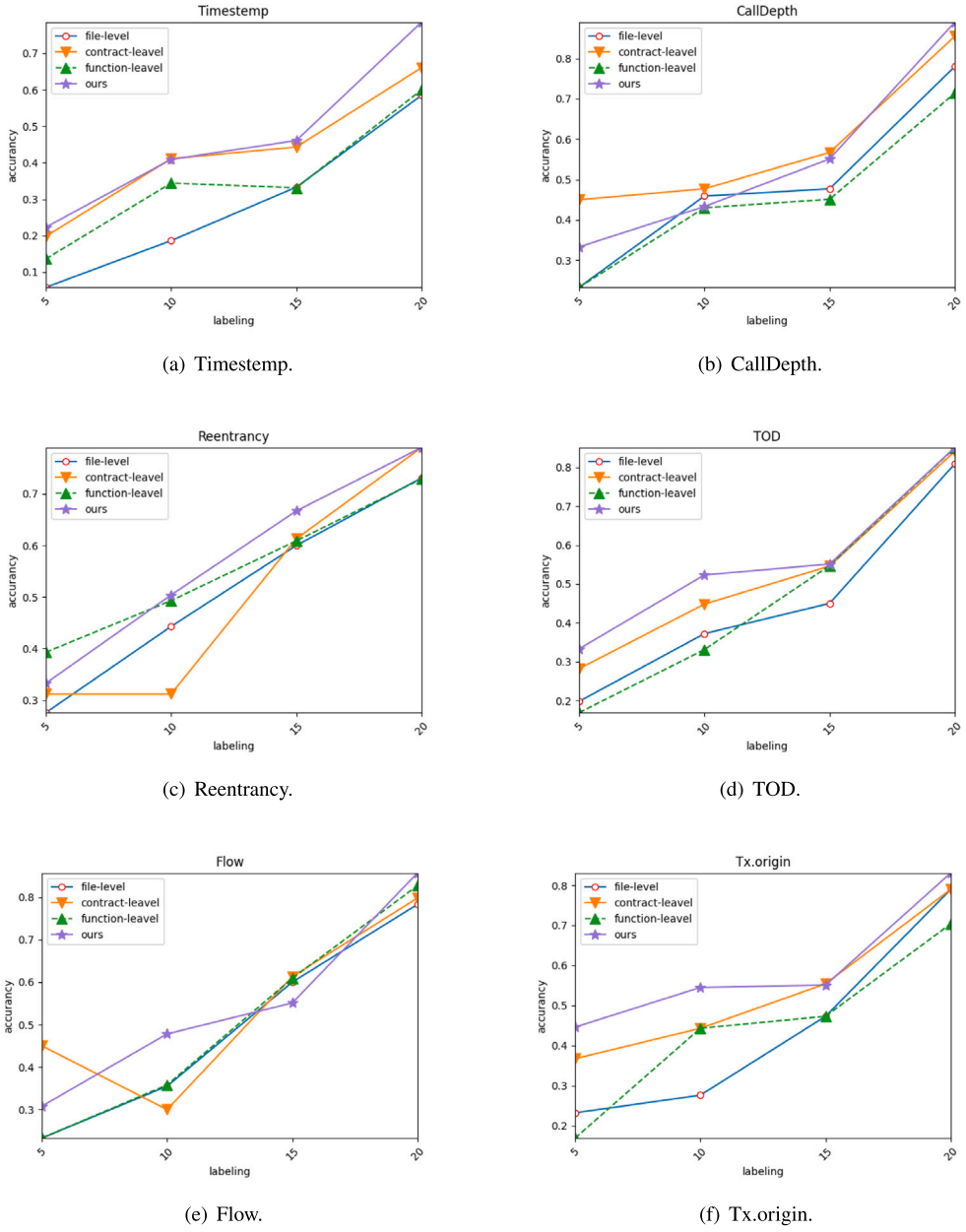
Measures	Labeling	Random	QBC	Density	EER	Unc
			%			
Accuracy	5	40.0	35.4	35.5	44.3	44.6
	10	47.4	55.4	47.3	50.5	54.5
	15	51.1	49.3	52.8	50.1	55.1
	20	74.0	77.3	84.3	74.7	83.1
Presion	5	17.9	12.9	11.0	12.4	19.4
	10	52.1	50.1	49.4	56.0	54.6
	15	59.5	59.7	59.5	62.2	65.1
	20	61.7	56.0	55.0	61.5	60.2
Recall	5	17.3	18.6	8.5	15.8	14.8
	10	52.5	60.0	49.3	51.4	56.0
	15	62.2	62.1	53.1	59.1	61.8
	20	71.9	73.1	75.3	73.5	76.3

the development document of Ethereum official programming language solidity, we summarized the following parts to be deleted from the source file, contract level and function level in the method section. In order to illustrate the impact of three-level data preprocessing on our framework, we used a single variable experiment to compare the experimental results of comprehensive data preprocessing and data preprocessing only at a certain level.

Fig. 2 shows the classification accuracy of *ASSBert* under different levels of data preprocessing on labeling from 5% to 20%. For the purpose of viewing the experimental results more intuitively, we use the line chart to view the accuracy changes in different data preprocessing labeling method when the labeling ratio are from 5% to 20%. With the rising proportion of labeling, the prediction accuracy of classification models for these six vulnerabilities under different levels of data processing shows the most significant growth trend as a whole. For the vuln1(Timestamp), when the labeling ratio is less than 15%, there is no obvious difference between our joint processing method and the contract level processing method, but it is significantly better than the other two methods. When the proportion of labeling is higher than 15%, our combined processing method has higher detection accuracy than the other three classification models(As shown in Fig. 2-(a)). For vuln2 (CallDepth), when the labeling ratio is less than 15%, the effect of contract level processing on model improvement is higher than the other three processing methods. However, with the increase of labeling to 20%, our combined processing method can achieve higher accuracy(As shown in Fig. 2-(b)). For vuln3 (Reentrancy), our combined processing method and function level processing method have significantly improved the accuracy of the model. Especially when the labeling ratio exceeds 10%, our combined processing method can make the model performance a greater improvement(As shown in Fig. 2-(c)). Unfortunately, for vuln4(TOD) and vuln5(Flow), our combined processing method does not perform well. When the labeling ratio is less than 15%, the improvement of model performance has no obvious advantage over the other three methods. And when the labeling ratio reached 20%, our combined processing method did not show better performance(The results are detailed in Fig. 2-(d,e)).

5. Discussion

The trained neural network can predict the unlabeled code data, but whether the model training is good or not determines whether the prediction result is correct. Vulnerability detection technology for smart contract is based on machine learning and needs a great deal of labeled data. Although the combination of active learning is be able to effectively increase the efficiency of manual labeling and save the manual labeling cost, the performance of model training only exploiting a small quantity of labeled data cannot achieve good results, or even worse. Inspiringly, we propose a new framework, called *ASSbert*, that levels active and semi upgraded bidirectional encoder representation from



(a) Timestemp. (b) CallDepth.

(c) Reentrancy. (d) TOD.

(e) Flow. (f) Tx.origin.

Fig. 2. Accuracy of ASSBert under different levels of data preprocessing on labeling from 5% to 20%. File-level, contract-level and method-level refer to data preprocessing only according to file level, contract level or method level. Our method is to conduct data preprocessing at three levels.

transformers network *ASSBert*. It collaboratively selects the valuable code data from the unlabeled set and adds them to the training set after manual or pseudo annotation. Furthermore, this can reliably improve the generalization performance of classifier models.

Applying semi-supervised learning directly to Bert will first provide a lot of pseudo tags to the unmarked code data. Training errors may continue to accumulate due to incorrect labeling. This can easily lead to model performance degradation. To solve this problem, we propose an active learning strategy. Our framework exploits using active learning to let experts label manually. It based on the uncertainty sampling strategy selects some *sol* files containing more information from unlabeled data sets and then merges them with the persent labeled data sets to build a training dataset. This enables the classification model to rapidly improve its generalization ability.

However, in the practical application of vulnerability detection for smart contracts, it is not necessary to mark all code data, which is time-consuming and costly. Based on this, we propose a mixed approach, which further combines the semi-supervised learning strategy effectively using the trained Bert for prediction to select some high-confidence unlabeled code data from the unlabeled dataset. To avoid the impact of bad tags, we reduce the amount of pseudo-label data relative to each loop. Each round of the active learning module in our framework needs to repeatedly query each sample in the sample pool, which will increase the response time of the framework, thus the model will cost more time. In the future work, we hope to access the samples based on the flow to reduce the running time of the model.

6. Conclusion

A major challenge for smart contract vulnerability detection is that most data in the current field are unlabeled data, and it is very difficult to obtain labeled data. Active learning can effectively reduce the workload of manual marking. However, the basic idea of active learning conflicts with model training in deep learning. The labeled set in classic active learning is not enough for model training of

classic deep learning. To cope with the problem that there are not enough labeled data from active learning for model training in deep learning. We proposed a novel framework that leverages active and semi-supervised bidirectional encoder representation from transformers network (*ASSBert*), which is dedicated to completing the task of smart contract vulnerability classification with a little of labeled code data and available unlabeled code data. In the framework we purposed, active learning selects highly uncertain code data from unlabeled sol files and adds them to the training set after manual annotation. Semi-supervised learning is used to continuously select a certain number of high-confidence unlabeled code data from unlabeled sol files, and put them into the training set after pseudo-labeling. By combining active learning and semi-supervised learning, we can get more valuable data to make the performance of the classification model better. And we choose to use the natural language processing model Bert as the classification model. Empirical evaluation of the dataset we collected included 6 vulnerabilities in about 20829 smart contracts demonstrating that our framework outperforms the baseline methods with a little of labeled code data and available unlabeled code data.

Moreover, it points out that our current work on vulnerability detection for smart contracts is not very satisfactory. The work of smart contract vulnerability detection must be more complex and challenging in reality. In order to solve more problems, we hope that the proposed framework can be expanded in future work. For the sake of saving running costs, in the future, we hope to combine stream-based active learning methods to shorten program execution time. From the perspective of data distribution, we will consider not only representative data but also data distribution in the future work. We plan to build a density estimator from the data distribution and expand the sampling strategy by integrating various aspects. In addition, we hope to make further breakthroughs in this area in the future.

CRediT authorship contribution statement

Xiaobing Sun: Methodology, Writing – original draft. **Liangqiong Tu:** Software. **Jiale Zhang:** Writing – review & editing, Methodology. **Jie Cai:** Formal analysis. **Bin Li:** Conceptualization, Supervision. **Yu Wang:** Writing – review & editing, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 62206238, No. 61972335, No. 61872312), Natural Science Foundation of Jiangsu Province, China (Grant No. BK20220562), Natural Science Foundation of Jiangsu Higher Education Institutions of China (Grant No. 22KJB520010), the Six Talent Peaks Project in Jiangsu Province, China (No. RJFW-053), the Jiangsu "333" Project, the Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University (No. KFKT2020B15, No. KFKT2020B16), the Future Network Scientific Research Fund Project (FNSRFP-2021-YB-47), the Yangzhou city-Yangzhou University Science and Technology Cooperation Fund Project (YZ2021157, YZ2021158), the Key Laboratory of Safety-Critical Software Ministry of Industry and Information Technology (No. NJ2020022), the Natural Science Research Project of Universities in Jiangsu Province (No. 20KJB520024), and Yangzhou University Top-level Talents Support Program (2019).

References

[1] [26] Cointegraph1. 2022, https://cointelegraph.com/ [Last Accessed 21 Mar 2022].
[2] Torres Christof Ferreira, Iannillo Antonio Ken, Gervais Arthur, State Radu. The eye of Horus: Spotting and analyzing attacks on ethereum smart contracts. 2021, arXiv preprint arXiv:2101.06204.
[3] Qu Youyang, Uddin Md Palash, Gan Chenquan, Xiang Yong, Gao Longxiang, Yearwood John. Blockchain-enabled federated learning: A survey. ACM Comput Surv 2022;55(4):1–35.
[4] Dao vulnerability. 2022, https://blog.ethereum.org/ [Last Accessed 21 July 2022].
[5] A postmortem on the parity multi-sig library self-destruc. 2022, https://www.parity.io/apostmortem-on-the-parity-multi-sig-libraryself-destruct/ [Last accessed 21 July 2022].
[6] Tikhomirov Sergei, Voskresenskaya Ekaterina, Ivanitskiy Ivan, Takhaviev Ramil, Marchenko Evgeny, Alexandrov Yaroslav. Smartcheck: Static analysis of ethereum smart contracts. In: Proceedings of the 1st international workshop on emerging trends in software engineering for blockchain. 2018, p. 9–16.
[7] Feist Josselin, Grieco Gustavo, Groce Alex. Slither: A static analysis framework for smart contracts. In: 2019 IEEE/ACM 2nd international workshop on emerging trends in software engineering for blockchain. IEEE; 2019, p. 8–15.
[8] Kalra Sukrit, Goel Seep, Dhawan Mohan, Sharma Subodh. Zeus: Analyzing safety of smart contracts. In: Ndss. 2018, p. 1–12.
[9] Park Daejun, Zhang Yi, Saxena Manasvi, Daian Philip, Roşu Grigore. A formal verification tool for ethereum VM bytecode. In: Proceedings of the 2018 26th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering. 2018, p. 912–5.
[10] Tsankov Petar, Dan Andrei, Drachsler-Cohen Dana, Gervais Arthur, Buenzli Florian, Vechev Martin. Securify: Practical security analysis of smart contracts. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. 2018, p. 67–82.
[11] Luu Loi, Chu Duc-Hiep, Olickel Hrishi, Saxena Prateek, Hobor Aquinas. Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 2016, p. 254–69.
[12] Jiang Bo, Liu Ye, Chan WK. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In: 2018 33rd IEEE/ACM international conference on automated software engineering. IEEE; 2018, p. 259–69.
[13] Zhang Jiale, Tu Liangqiong, Cai Jie, Sun Xiaobing, Li Bin, Chen Weitong, et al. Vulnerability detection for smart contract via backward Bayesian active learning. In: International conference on applied cryptography and network security. Springer; 2022, p. 66–83.
[14] Zhdanov Fedor. Diverse mini-batch active learning. 2019, arXiv preprint arXiv:1901.05954.
[15] Tran Toan, Do Thanh-Toan, Reid Ian, Carneiro Gustavo. Bayesian generative active deep learning. In: International conference on machine learning. PMLR; 2019, p. 6295–304.
[16] Wang Keze, Zhang Dongyu, Li Ya, Zhang Ruimao, Lin Liang. Cost-effective active learning for deep image classification. IEEE Trans Circuits Syst Video Technol 2016;27(12):2591–600.
[17] Desai Shasvat, Ghose Debasmita. Active learning for improved semi-supervised semantic segmentation in satellite images. In: Proceedings of the IEEE/CVF winter conference on applications of computer vision. 2022, p. 553–63.
[18] Hossain HM Sajjad, Roy Nirmalya. Active deep learning for activity recognition with context aware annotator selection. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. 2019, p. 1862–70.
[19] Siméoni Oriane, Budnik Mateusz, Avrithis Yannis, Gravier Guillaume. Rethinking deep active learning: Using unlabeled data at model training. In: 2020 25th international conference on pattern recognition. IEEE; 2021, p. 1220–7.
[20] Brent Lexi, Jurisevic Anton, Kong Michael, Liu Eric, Gauthier Francois, Gramoli Vincent, et al. Vandal: A scalable security analysis framework for smart contracts. 2018, arXiv preprint arXiv:1809.03981.
[21] Torres Christof Ferreira, Schütte Julian, State Radu. Osiris: Hunting for integer bugs in ethereum smart contracts. In: Proceedings of the 34th annual computer security applications conference. 2018, p. 664–76.
[22] Liu Chao, Liu Han, Cao Zhao, Chen Zhong, Chen Bangdao, Roscoe Bill. Reguard: finding reentrancy bugs in smart contracts. In: 2018 IEEE/ACM 40th international conference on software engineering: companion. IEEE; 2018, p. 65–8.
[23] Kevin N'DA Aboua Ange, Matalonga Santiago, Dahal Keshav. Applicability of the software security code metrics for ethereum smart contract. In: The international conference on deep learning, big data and blockchain. Springer; 2021, p. 106–19.
[24] Momeni Pouyan, Wang Yu, Samavi Reza. Machine learning model for smart contracts security analysis. In: 2019 17th international conference on privacy, security and trust. IEEE; 2019, p. 1–6.
[25] Liao Jian-Wei, Tsai Tsung-Ta, He Chia-Kang, Tien Chin-Wei. Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing. In: 2019 sixth international conference on internet of things: systems, management and security. IEEE; 2019, p. 458–65.

[26] Qian Peng, Liu Zhenguang, He Qinming, Zimmermann Roger, Wang Xun. Towards automated reentrancy detection for smart contracts based on sequential models. IEEE Access 2020;8:19685–95.
[27] Ashizawa Nami, Yanai Naoto, Cruz Jason Paul, Okamura Shingo. Eth2Vec: Learning contract-wide code representations for vulnerability detection on ethereum smart contracts. In: Proceedings of the 3rd ACM international symposium on blockchain and secure critical infrastructure. 2021, p. 47–59.
[28] Mi Feng, Wang Zhuoyi, Zhao Chen, Guo Jinghui, Ahmed Fawaz, Khan Latifur. VSCL: Automating vulnerability detection in smart contracts with deep learning. In: 2021 IEEE international conference on blockchain and cryptocurrency. IEEE; 2021, p. 1–9.
[29] Huang Sheng-Jun, Jin Rong, Zhou Zhi-Hua. Active learning by querying informative and representative examples. Adv Neural Inf Process Syst 2010;23.
[30] Atighehchian Parmida, Branchaud-Charron Frédéric, Lacoste Alexandre. Bayesian active learning for production, a systematic study and a reusable library. 2020, arXiv preprint arXiv:2006.09916.
[31] Tsymbalov Evgenii, Makarychev Sergei, Shapeev Alexander, Panov Maxim. Deeper connections between neural networks and Gaussian processes speed-up active learning. 2019, arXiv preprint arXiv:1902.10350.
[32] Kirsch Andreas, Van Amersfoort Joost, Gal Yarin. Batchbald: Efficient and diverse batch acquisition for deep Bayesian active learning. Adv Neural Inf Process Syst 2019;32.
[33] Cakmak Maya, Thomaz Andrea L. Eliciting good teaching from humans for machine learners. Artificial Intelligence 2014;217:198–215.
[34] Donmez Pinar, Carbonell Jaime G, Schneider Jeff. Efficiently learning the accuracy of labeling sources for selective sampling. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining. 2009, p. 259–68.
[35] Zhang Xiao-Yu, Wang Shupeng, Yun Xiaochun. Bidirectional active learning: A two-way exploration into unlabeled and labeled data set. IEEE Trans Neural Netw Learn Syst 2015;26(12):3034–44.
[36] Luo Guangchun, Ma Ying, Qin Ke. Active learning for software defect prediction. IEICE Trans Inform Syst 2012;95(6):1680–3.
[37] Lu Huihua, Cukic Bojan. An adaptive approach with active learning in software fault prediction. In: Proceedings of the 8th international conference on predictive models in software engineering. 2012, p. 79–88.
[38] Lu Huihua, Kocaguneli Ekrem, Cukic Bojan. Defect prediction between software versions with active learning and dimensionality reduction. In: 2014 IEEE 25th international symposium on software reliability engineering. IEEE; 2014, p. 312–22.
[39] Xu Zhou, Liu Jin, Luo Xiapu, Zhang Tao. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. In: 2018 IEEE 25th international conference on software analysis, evolution and reengineering. IEEE; 2018, p. 209–20.
[40] Arazo Eric, Ortego Diego, Albert Paul, O'Connor Noel E, McGuinness Kevin. Pseudo-labeling and confirmation bias in deep semi-supervised learning. In: 2020 international joint conference on neural networks. IEEE; 2020, p. 1–8.
[41] Wang Lei, Qian Qing, Zhang Qiang, Wang Jishuai, Cheng Wenbo, Yan Wei. Classification model on big data in medical diagnosis based on semi-supervised learning. Comput J 2022;65(2):177–91.
[42] Yalniz I Zeki, Jégou Hervé, Chen Kan, Paluri Manohar, Mahajan Dhruv. Billion-scale semi-supervised learning for image classification. 2019, arXiv preprint arXiv:1905.00546.
[43] Taherkhani Fariborz, Kazemi Hadi, Nasrabadi Nasser M. Matrix completion for graph-based deep semi-supervised learning. In: Proceedings of the AAAI conference on artificial intelligence, Vol. 33, no. 01. 2019, p. 5058–65.
[44] Solidity. 2022, https://docs.soliditylang.org/en/v0.8.16/ [Last Accessed 5 Sep 2022].
[45] Yu Zhe, Theisen Christopher, Williams Laurie, Menzies Tim. Improving vulnerability inspection efficiency using active learning. IEEE Trans Softw Eng 2019.
[46] DASP TOP 10. 2022, https://dasp.co/ [Last Accessed 21 Mar 2022].
[47] Durieux Thomas, Ferreira João F, Abreu Rui, Cruz Pedro. Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering. 2020, p. 530–41.
[48] SoliAudit vulnerability analyzer dataset. 2022, https://goo.gl/UAUpK5/ [Last Accessed 21 Mar 2022].
[49] Ghaleb Asem, Pattabiraman Karthik. How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection. In: Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis. 2020, p. 415–27.
[50] Devlin Jacob, Chang Ming-Wei, Lee Kenton, Toutanova Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018, arXiv preprint arXiv:1810.04805.
[51] Abe Naoki. Query learning strategies using boosting and bagging. In: Proc. of 15ˆ< th> int. cmf. on machine learning. 1998, p. 1–9.
[52] Roy Nicholas, McCallum Andrew. Toward optimal active learning through monte carlo estimation of error reduction. In: ICML, Vol. 2. Williamstown; 2001, p. 441–8.