Improving Smart Contract Security with Contrastive Learning-based Vulnerability Detection

Yizhou Chen
Key Lab of HCST (PKU), MOE;
School of Computer Science, Peking University
Beijing, China
yizhouchen@stu.pku.edu.cn

Zeyu Sun*
Science & Technology on Integrated Information System
Laboratory, Institute of Software, Chinese Academy of
Sciences
Beijing, China
szy_@pku.edu.cn

Zhihao Gong
Key Lab of HCST (PKU), MOE;
School of Computer Science, Peking University
Beijing, China
zhihaogong@stu.pku.edu.cn

Dan Hao
Key Lab of HCST (PKU), MOE;
School of Computer Science, Peking University
Beijing, China
haodan@pku.edu.cn

ABSTRACT

Currently, smart contract vulnerabilities (SCVs) have emerged as a major factor threatening the transaction security of blockchain. Existing state-of-the-art methods rely on deep learning to mitigate this threat. They treat each input contract as an independent entity and feed it into a deep learning model to learn vulnerability patterns by fitting vulnerability labels. It is a pity that they disregard the correlation between contracts, failing to consider the commonalities between contracts of the same type and the differences among contracts of different types. As a result, the performance of these methods falls short of the desired level.

To tackle this problem, we propose a novel Contrastive Learning Enhanced Automated Recognition Approach for Smart Contract Vulnerabilities, named Clear. In particular, Clear employs a contrastive learning (CL) model to capture the fine-grained correlation information among contracts and generates correlation labels based on the relationships between contracts to guide the training process of the CL model. Finally, it combines the correlation and the semantic information of the contract to detect SCVs. Through an empirical evaluation of a large-scale real-world dataset of over 40K smart contracts and compare 13 state-of-the-art baseline methods. We show that Clear achieves (1) optimal performance over all baseline methods; (2) 9.73%-39.99% higher F1-score than existing deep learning methods.

CCS CONCEPTS

• **Security and privacy** → **Software security engineering**; **Software security engineering**; • **Computing methodologies** → **Knowledge representation and reasoning**.

KEYWORDS

Smart contract, Vulnerability detection, Deep learning, Contrastive learning

1 INTRODUCTION

In contemporary times, transactions based on blockchain systems and smart contracts are becoming increasingly popular in both personal and commercial settings [22, 30]. Yet, this increased reliance on blockchain systems has made them a tempting target for cybercriminals seeking to exploit software vulnerabilities for illegal financial gain [1, 28]. With the exponential growth of the virtual currency market, Smart Contract Vulnerabilities (SCVs) have become a major risk threatening secure transactions on the blockchain. The potential exploitation of these vulnerabilities by malicious actors could result in the compromise of virtual assets, putting users at risk of significant financial losses [25, 35]. In 2016, the Decentralized Autonomous Organization on Ethereum was attacked, and the attacker exploited an SCV to steal approximately $50 million worth of ether [4]. Moreover, in 2018, the decentralized exchange Bancor suffered an SCV, resulting in the theft of roughly $23.5 million worth of cryptocurrencies. The above emergencies reveal that Smart Contract Vulnerability Detection (SCVD) has become an urgent task.

To detect SCVs, numerous researchers proposed effective methods, which are broadly divided into two categories. The first line of work [6, 21, 23, 24, 32–34] is rule-based techniques, which identify SCVs through predefined rules or manually-defined patterns on
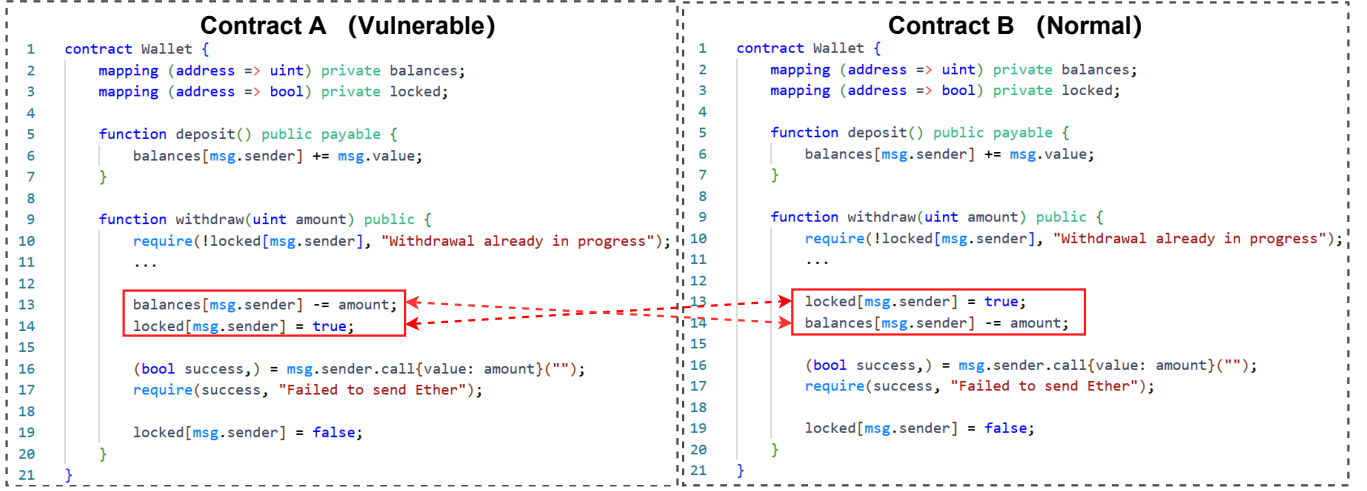
Figure 1: Two examples of smart contracts.

the smart contract code and its execution. However, the vulnerabilities that exist in smart contracts are diverse, which can make predefined patterns insufficient in covering all possible vulnerability types. As a result, these methods may produce false positives or false negatives, which undermine their effectiveness in detecting vulnerabilities accurately [26]. Moreover, developing these patterns is a time-consuming and error-prone process that relies heavily on manual work. Therefore, the researchers recognize the need to explore alternative approaches that can help reduce the cost and improve the accuracy of SCVD.

Another line of work [15, 18, 26, 42] utilizes deep learning methods to automatically detect SCVs, resulting in impressive performance gains. These methods use neural networks to learn the vulnerability patterns and detect SCVs. The commonality of these methods is that they treat each input contract as an isolated entity labeled with whether it is vulnerable. Indeed, a contract usually contains many lines of code, but only a few are relevant to SCVs. Some fine-grained information can hardly be learned by the existing methods, but can be caught by the difference between the vulnerable and non-vulnerable contracts. In other words, these deep learning methods have achieved limited performance because they ignore the correlation between contracts, including the difference between vulnerable and non-vulnerable contracts, as well as the commonalities between vulnerable contracts.

To solve the problem, we propose a novel Contrastive Learning Enhanced Automated Recognition approach for SCVs, called Clear. Clear introduces the contract correlation into the field of SCVD and leverages the contrastive learning (CL) model to learn pairwise comparisons of smart contracts and find their correlations. In addition, we guide the training process of the CL model by reusing existing vulnerability labels to generate correlation labels. These efforts are used to improve the performance of SCVD. To outline briefly, Clear samples pairs of contracts from the dataset and generate a correlation label for the contract pairs. Then, a CL model is constructed, which consists of a contextual augmentation module, a Transformer module, and a contrastive loss function,

to learn the fine-grained correlation information between pairs of contracts by fitting correlation labels. Finally, we fine-tune the Transformer module and fit vulnerability labels to enhance the performance of vulnerability detection.

Our proposed method has been rigorously evaluated on the largest established dataset on SCVD, which consists of over 40K real-world smart contracts, by comparing it against 13 state-of-the-art SCVD methods. The quantitative experimental results show the proposed Clear outperforms all the state-of-the-art methods across all metrics. In particular, Clear achieves significant improvement over even the best-performing method DMT [26]: precision increased from 87.28% to 93.64% (by 7.29%), recall improved from 85.13% to 95.44% (by 12.11%), and F1-score elevated from 86.14% to 94.52% (by 9.73%) on three types of SCVs, averagely. Besides, our ablation study shows that Clear achieves outperformance by clustering vulnerability contracts in the feature space and separating them from non-vulnerability contracts. Moreover, we experimentally demonstrate that the proposed CL model enhances RNN-based models (i.e., RNN, LSTM, GRU) and boosts their performance by 40.51%-50.94% in terms of the F1-score compared to the original model. In summary, this paper makes the following contributions.

- **A contrastive-learning-based vulnerability detection technique Clear**, which utilizes fine-grained correlation information among smart contracts to improve the performance of SCVD.
- **An extensive experiment** on a large-scale dataset of over 40,000 smart contracts comparing against 13 state-of-art SCV methods, which shows the effectiveness of Clear.
- **A reproducible package** available at https://github.com/chenpp1881/Clear.

2 MOTIVATING EXAMPLES

Smart contract development is a relatively unfamiliar field, and numerous developers may lack an in-depth understanding of smart contract security. They may focus on the functionality and business logic of the contract while ignoring potential security risks, thus
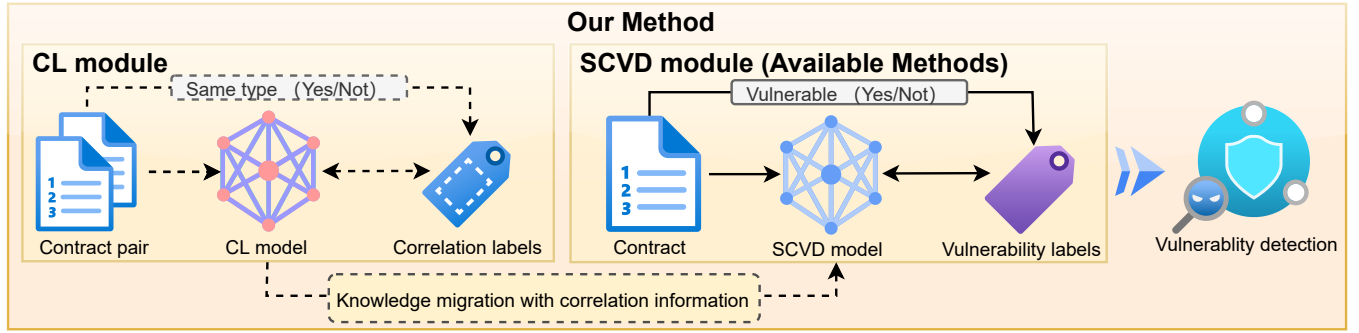
Figure 2: Architecture of our method and the available methods.

introducing vulnerabilities in the code-writing process. Moreover, developers often unknowingly make small mistakes that can result in vulnerabilities. As a result, contracts that exhibit vulnerabilities can closely resemble those that do not. Figure 1 shows an example where both contracts have identical functionality, with the only distinction residing in the order of statements within the "withdraw" function (lines 13-14). In *Contract B*, the account is initially locked (line 13), followed by the updating and transfer of the account balance (lines 14-17). Conversely, in *Contract A*, the account balance is updated (line 13) before the account is locked (line 14). This minor difference in *Contract A* introduces a reentrancy vulnerability[1]. Specifically, when an attacker first calls the "withdraw" function, the statement "balances[msg.sender] -= amount" may cause "balances[msg.sender]" to become a negative value. However, since "locked[msg.sender]" has not been set to true yet, the attacker can still re-enter the "withdraw" function. On the second entry into the "withdraw" function, due to "balances[msg.sender]" being a negative value, the attacker may successfully execute additional malicious operations, resulting in a reentrancy attack. In practical development scenarios, the logic of smart contracts is often highly complex, leading to the frequent occurrence of the aforementioned situation. Unfortunately, current SCVD methods treat smart contracts as isolated entities and rely on deep learning models to learn vulnerability features or patterns to identify SCVs. These methods may overlook vulnerabilities triggered by subtle faults. Specifically, the limitations of existing SCVD methods stem from their architecture, which requires deep learning models to independently explore and learn semantic information from the entire contract code, guided by vulnerability labels, to identify possible vulnerability patterns. This architecture lacks sufficient detail for deep learning models to adequately comprehend the fundamental nature of vulnerabilities. The challenges faced by deep learning models in accurately capturing correlation information among contracts under this architecture encompass fine-grained differences between vulnerable and non-vulnerable contracts, as well as commonalities among vulnerable contracts. Undoubtedly, correlation information plays a crucial role in effectively identifying SCVs.

However, until now, the impact of contract correlations on SCVD is still unexplored in existing studies. The CL model provides important inspiration for our work. The CL model was originally developed as an unsupervised learning technique in the field of computer vision to learn representations by uncovering the underlying similarities and dissimilarities among samples [2, 5, 14, 38]. Therefore, to address the aforementioned issues, we extend it to the SCVD tasks of supervised learning by adapting the methodology used for constructing sample pairs and correlation labels. To be specific, as shown in Figure 2, our method diverges from existing SCVD methods, as we initially leverage the CL model to learn the correlations among contracts. Subsequently, we migrate the correlation knowledge to the SCVD model, integrating it with the vulnerability features of the contracts to accurately detect and identify SCVs. It is worth emphasizing that our method is specifically designed to target common and subtle faults in SCVs. If similar vulnerability characteristics exist in the software of other domains, our method may also be adapted to detect them.

3 METHODOLOGY

3.1 Method Overview

In migrating the CL model to the SCVD domain, we are faced with three primary issues. (1) It is essential to determine an appropriate label that can guide the CL model in learning effective correlation information. (2) The neural network for contextual semantic representation of smart contracts needs to be refined and designed to improve the generalization of the CL model. (3) Leveraging both correlation information and vulnerability features to enhance the performance of SCVD.

Therefore, we present a novel approach for SCV automated detection, named Clear, as illustrated in Figure 3. Clear sequentially tackles the aforementioned issues through three steps:

1. **Data Sampling:** contract pairs are sampled from the dataset to serve as input for the CL model. A correlation label is assigned to each pair, indicating their relationship and guiding the training process of the CL model.
2. **Contrastive Learning:** we devise a CL model that incorporates a contextual augmentation module and a Transformer-based feature learning module. By computing contrastive loss, the model learns to capture correlations between contract pairs that align with the provided labels.

---

[1]Reentrancy vulnerability is one of the most common and dangerous vulnerabilities in smart contracts, whose details are given in Section 4.1.

3. **Vulnerability Detection:** we fine-tune the Transformer model from Stage 2 and combine the outputs of the CL model to recognize SCVs by a fully connected neural network.

3.2 Data Sampling

For the CL module, we incorporate correlation labels to guide the training process. These labels are constructed based on the relationships between sampled contracts. Therefore, employing a suitable sampling strategy is crucial as it can greatly enhance the performance by ensuring the utilization of high-quality sample pairs for training, while minimizing the introduction of bias into the learning process. Motivated by this, our sampling strategy is as follows:

There are three types of relationships for contract pairs, i.e. "**V-V**", "**N-N**", and "**V-N**", where **V** and **N** denote **V**ulnerable and **N**on-vulnerable contracts, respectively. Our intuition is that the relationships of "**V-V**" and "**V-N**" are more important, because we would like to discover the commonality in "**V-V**" and differences in "**V-N**" by CL. In contrast, the "**N-N**" is not substantially helpful in identifying SCV. Therefore, our sampling strategy is to extract all vulnerable contracts from the original dataset and create a new set called the POS set. Then, for each contract in the original dataset, we randomly select a contract from the POS set to form a pair of contracts as input for the CL model. It should be noted that this sampling strategy does not have "**N-N**" relationship. Finally, the correlation labels L_{CL} of the contract pairs are constructed to guide the training of the CL model. The rule is as follows:

$$L_{CL} = \begin{cases} 1, & \text{If "V-V"} \\ 0, & \text{If "V-N"} \end{cases}. \tag{1}$$

3.3 Contrastive Learning

To better model correlations, in the CL stage, we first apply contextual augmentation to the input contract, then use a Transformer to learn contract features, and finally employ the contrastive loss function to optimize the Transformer model. It should be noted that both contracts in a contract pair undergo the same encoding process and share identical model parameters. Therefore, for simplicity and clarity, we only demonstrate the encoding process for a single contract in the following subsections.

3.3.1 Contextual Augmentation. To enhance the comprehension of semantic and structural features in the contract code, as well as to facilitate understanding of contract correlation, we incorporate a contextual augmentation module, i.e., masked language model (MLM), at the beginning of the CL stage. The core of MLM is a self-supervised Transformer model. In simple terms, MLM involves randomly masking certain tokens in the input data. The Transformer model is then tasked with predicting the masked tokens based on the surrounding context. Predicting the masked tokens prompts the model to discern meaningful patterns, relationships, and dependencies within the contract code, facilitating a more comprehensive understanding of its underlying structure and semantics. This approach helps the model capture important features and contextual information, which can prove advantageous for downstream prediction tasks.

To be specific, given a code sequence, we randomly select 30% of the tokens to be replaced with a special [Mask] token, and keep the remaining 70% unchanged. Then, the Transformer is utilized to predict the original token that corresponds to the masked token. The loss function of an MLM can be represented using the cross-entropy loss function as follows:

$$\text{Loss}_{MLM} = -\sum_{j \in T} y_j \log \hat{y}_j, \tag{2}$$

where T denotes a set of the index of masked tokens, y_j is the ground truth label of the j-th masked token, and $\hat{y}_j$ is the predicted probability of the j-th masked token being the ground truth label. Specifically, in an MLM, only the tokens that are replaced with the special token [MASK] are used to calculate the loss.

3.3.2 Feature Learning. Our feature learning module also follows a standard Transformer process with a minor modification. Given that the CL module necessitates the computation of distances in the complete sequence representation of two samples, we incorporate CLS vectors as a way to extract the entire semantic information of code samples. This idea is inspired by previous research [7]. Incidentally, the introduction of CLS vectors in the CL stage can enhance model efficiency by eliminating additional processes, such as sequence modeling and data alignment, solely contrasting two CLS vectors. The $\textbf{\textit{CLS}} \in \mathbb{R}^{1 \times k}$ vector serves as an additional input token and is generated in the following manner:

$$\textbf{\textit{CLS}} = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} X_i', \tag{3}$$

where $X' \in \mathbb{R}^{n \times k}$ is the output X of MLM, k is the word embedding dimension. Then, the position encoding PE is employed to furnish token-level positional information for X. The specific process is as follows:

$$PE_{(pos,2l)} = \sin \frac{pos}{10000^{2l/k}}, \tag{4}$$

$$PE_{(pos,2l+1)} = \cos \frac{pos}{10000^{2l/k}}, \tag{5}$$

where pos is the position identifier that records the position information of the token in the sequence, and l denotes the dimension index.

Subsequently, together with CLS and X', it serves as input for the multi-head attention mechanism (MHAM). The mathematical process can be represented as:

$$\textbf{\textit{CLS}}', F = \text{MHAM}(\textbf{\textit{CLS}} \oplus (X' + PE)), \tag{6}$$

where $\oplus$ and $+$ denote concatenation and element-wise addition.

Whereafter, the $\textbf{\textit{CLS}}'$ vectors capture global semantic information about the contract and serve as representative summaries of its overall content. They are utilized in the CL stage to establish correlations between instances of contracts. Conversely, the feature vectors $F \in \mathbb{R}^{n \times k}$, which comprise the encoded representations of each token and its contextual dependencies, are utilized in the vulnerability detection stage.
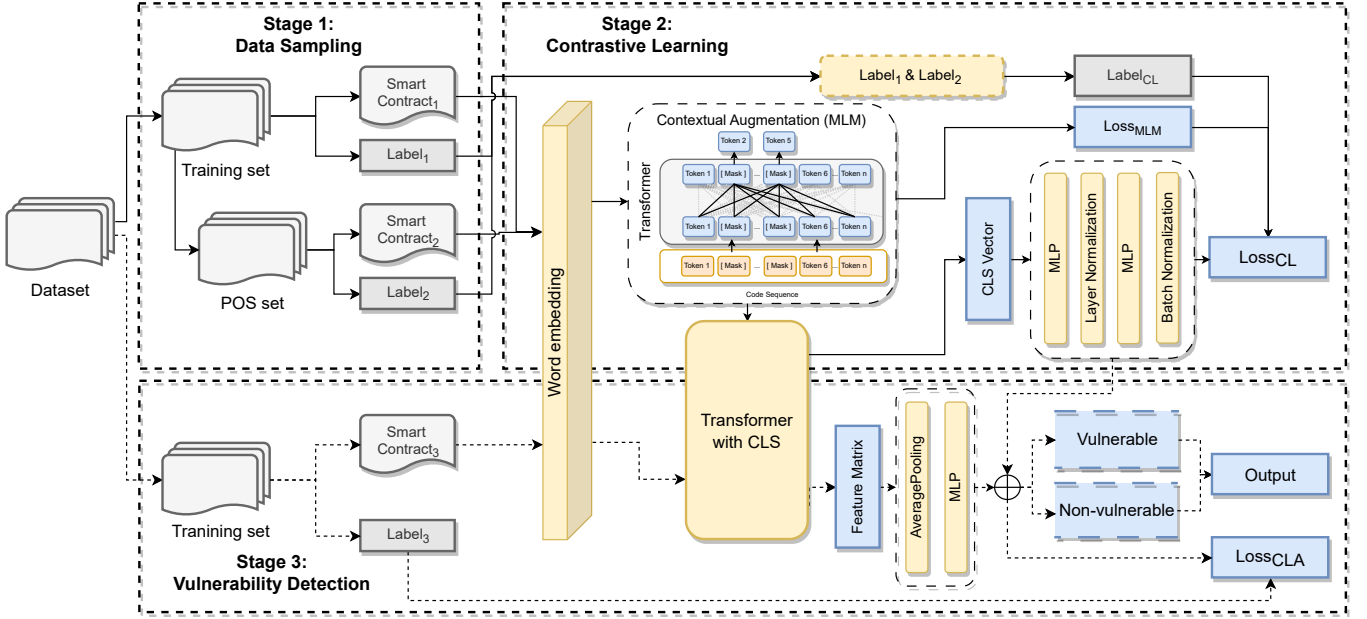
Figure 3: Overview of Clear, which encompasses both the CL process, depicted by solid lines indicating the data flow, and the subsequent vulnerability detection process, represented by dotted arrows indicating the data flow.

3.3.3 Contrastive Loss. We then apply two linear transformations - L2 normalization and batch normalization - to process the CLS'. Specifically, L2 normalization promotes a more balanced distribution of the vector's elements, preventing any single feature from dominating the learning process. Batch normalization improves model convergence and stability by reducing internal covariate shift, which is the variation in activation distribution across different layers during training. Their mathematical process is as follows:

$$v = \text{BatchNorm}(W_2 \cdot \text{LayerNorm}(W_1 \cdot CLS')), \tag{7}$$

where W_1 and W_2 are weights of the linear transformation and v is the ultimate global vector representation of a contract in the CL stage.

In this way, the contract pair yields a pair of vector representations $[v_a, v_b]$. Then, we compute the contrastive loss $Loss_{CL}$ with the correlation label L_{CL}^{ab}. The contrastive loss is formulated as follows:

$$\begin{aligned}\text{Loss}_{CL}(v_a, v_b, L_{CL}^{ab}) = &\ L_{CL}^{ab} \cdot \text{sim}(v_a, v_b)^2 \\ &+ (1 - L_{CL}^{ab}) \cdot \max(0, M - \text{sim}(v_a, v_b))^2, \end{aligned} \tag{8}$$

where $\text{sim}(\cdot)$ represents the euclidean distance of the two vectors, and M is the margin that determines the threshold for dissimilarity.

Finally, the contrastive loss, denoted as Loss_{CL}, and the MLM loss, denoted as Loss_{MLM}, are combined to optimize the model. The overall loss function can be expressed as follows:

$$\text{Total Loss} = \lambda_{CL} \cdot \text{Loss}_{CL} + \lambda_{MLM} \cdot \text{Loss}_{MLM}.$$

In this equation, λ_{CL} and λ_{MLM} are hyperparameters that control the relative importance of the contrastive loss and MLM loss, respectively.

3.4 Vulnerability Detection

During the vulnerability detection stage, we focus on fine-tuning the Transformer encoder mentioned earlier to accurately detect SCV. After obtaining the feature representation F of the contracts, we combine F and correlation features v to detect SCV using a fully connected neural network. The process can be represented as follows:

$$\hat{L} = \sigma(W_3 \cdot (\text{AvgPooling}(F) \oplus v) + b), \tag{9}$$

where σ is the sigmoid activation function, W_3 is the weight matrix and b is the bias vector, $\oplus$ indicates concatenation operation. The output $\hat{L}$ is the predicted probability (vulnerable or non-vulnerable) and calculates the loss with the real vulnerability label $\hat{y} \in (0, 1)$ by the cross entropy loss Loss_{CLA}.

$$\text{Loss}_{CLA}(\hat{L}, \hat{y}) = -\left(\hat{y}\log\left(\hat{L}\right) + (1 - \hat{y})\log\left(1 - \hat{L}\right)\right). \tag{10}$$

4 EXPERIMENTAL SETTINGS

We conduct comprehensive evaluations of our proposed framework to address the following Research Questions (RQ):

RQ1: How does our method Clear perform compared against 13 state-of-the-art SCVD techniques?

RQ2: How do the different modules affect the performance of the proposed approach?

RQ3: Does our proposed CL module enhance the performance of other deep learning models besides Transformer?

4.1 Dataset

We select the largest publicly available vulnerability dataset for smart contracts [26], which consists of 40K real-world smart contracts. The dataset is carefully labeled with distinct types of SCVs.

Table 1: The performance evaluation of our method is compared with 13 baseline models involving baselines in terms of Recall (R), Precision (P) and F1-score (F). "n/a" denotes not applicable.

Line #	Methods	RE			TD			IO			Average		
		R(%)	P(%)	F(%)	R(%)	P(%)	F(%)	R(%)	P(%)	F(%)	R(%)	P(%)	F(%)
1	sFuzz	14.95	10.88	12.59	27.01	23.15	24.93	47.22	58.62	52.31	29.73	30.88	29.94
2	Smartcheck	16.34	45.71	24.07	79.34	47.89	59.73	56.21	45.56	50.33	50.63	46.39	44.71
3	Osiris	63.88	40.94	49.90	55.42	59.26	57.28	n/a	n/a	n/a	n/a	n/a	n/a
4	Oyente	63.02	46.56	53.55	59.97	61.04	59.47	n/a	n/a	n/a	n/a	n/a	n/a
5	Mythril	75.51	42.86	54.68	49.80	57.50	53.37	62.07	72.30	66.80	62.46	57.55	58.28
6	Securify	73.06	68.40	70.41	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7	Slither	73.50	74.44	73.97	67.17	69.27	68.20	52.27	70.12	59.89	64.31	71.28	67.35
8	GCN	73.18	74.47	73.82	77.55	74.93	76.22	69.74	69.01	69.37	73.49	72.80	73.14
9	TMP	75.30	76.04	75.67	76.09	78.68	77.36	70.37	68.18	69.26	73.92	74.30	74.10
10	AME	78.45	79.62	79.03	80.26	81.42	80.84	69.40	70.25	69.82	76.04	77.10	76.56
11	SMS	77.48	79.46	78.46	91.09	89.15	90.11	73.69	76.97	75.29	80.75	81.86	81.29
12	DMT	81.06	83.62	82.32	96.39	93.60	94.97	77.93	84.61	81.13	85.13	87.28	86.14
13	LineVul	73.01	85.19	78.63	67.46	89.47	76.92	74.20	74.10	74.15	71.56	82.92	76.57
14	Clear	96.43	96.81	96.62	98.41	94.30	96.31	91.48	89.81	90.64	95.44	93.64	94.52

Among the 40K contracts, 4290 contracts were identified to contain vulnerabilities: 680 contracts were identified to possess reentrancy vulnerabilities (RE), 2242 contracts exhibited timestamp dependency vulnerabilities (TD), and 1368 contracts were found to have integer overflow/underflow vulnerabilities (IO). This dataset randomly assigns 80% contracts as the training set and the remaining 20% as the test set, and in our evaluation, we use the same split sets.

These vulnerabilities are widely studied in prior work [26], because a large portion of the financial loss in Ethereum is attributed to these vulnerabilities [12, 21, 23, 32, 34] and existing research has demonstrated that these vulnerabilities are prevalent in Ethereum smart contracts. In particular, **RE** occurs when a contract allows an external attacker to re-enter a function before the previous execution completes, leading to unexpected and unauthorized behavior [10]. **TD** arises from the reliance on block timestamps for critical decisions within smart contracts. Attackers can manipulate timestamps to their advantage, compromising contract logic and enabling activities such as front-running or denial-of-service attacks [21]. **IO** occurs when arithmetic operations on integers exceed the maximum or minimum representable values [13]. These vulnerabilities can result in incorrect calculations, allowing attackers to manipulate values, bypass security checks, and cause financial harm.

4.2 Baselines

In our evaluation, we first select a set of baseline methods specifically designed for SCVD. These baselines represent state-of-the-art approaches in the SCVD field. They can be broadly classified into two categories: rule-based techniques and neural network-based techniques.

Rule-based techniques, as the category of baselines, rely on rule-based heuristics to identify SCVs, including sFuzz [24], Smartcheck [32], Osiris [33], Oyente [21], Mythril [23], Securify [34], and Slither [6].

Neural network-based methods, on the other hand, leverage deep learning techniques to identify SCVs. Here, we include five state-of-art neural-learning-based SCVD methods, including GCN [15], TMP [42], AME [18], SMS [26], and DMT [26].

General vulnerability detection methods [8, 17, 40] exist, but none of them yields satisfactory results in the field of SCVD. Therefore, in this paper, we have chosen only the best-performing one, namely LineVul [8], as the representative of general methods.

The details of all baselines are in Section 6.

4.3 Metrics

Following prior work [18, 19, 26], we use three common evaluation metrics to assess the performance of the SCVD methods, which are precision, recall, and F1 score. Given true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) of a classification model, precision, recall, and F1 score are defined below.

Precision: The proportion of correctly predicted positive instances among all instances predicted as positive. It is calculated as $P = \frac{TP}{TP+FP}$.

Recall: The proportion of correctly predicted positive instances among all actual positive instances. It is calculated as $R = \frac{TP}{TP+FN}$.

F1-score: The harmonic mean of precision and recall, which combines both metrics to provide a single measure of classification performance. It is calculated as $F1 = 2 \times \frac{precision \times recall}{precision+recall}$.

4.4 Implementation Details

For the hyperparameters of the experiment, the dimensionality of the word embeddings is set to 512. The Transformer model utilizes a six-layer multi-head attention layer with eight attention heads. During training, the learning rate is initialized to 1 x 10^{-5} and is optimized using the AdamW optimizer [20]. The batch size is fixed at eight. The total losses in the CL stage include contrastive loss Loss_{CL} and MLM loss Loss_{MLM}, where we set the weights for them, λ_{CL} and λ_{MLM}, to be 1.0 and 0.1 respectively. For all the baselines, we utilize the complete code from their provided opensource libraries and adhere to the configurations specified in their research. During the CL stage, we set the number of training epochs to 100. In the vulnerability detection stage, we perform training for 20 epochs to fine-tune the model and output the result of the last epoch. The aforementioned procedure is iterated five times and the average value is chosen as the conclusive outcome of this study.

The experiments are conducted using hardware resources that included 2 Nvidia RTX 3090 GPUs with 48GB video memory. These GPUs are utilized in parallel for training. For the software environment, we employ Ubuntu 20.04 LTS as the operating system.

5 RESULTS

5.1 RQ1: Effectiveness of the Clear

Table 1 shows the performance evaluation of 13 SCVD methods, focusing on three prevalent and critical vulnerabilities: RE, TD, and IO. We use bold font to represent the best result of all compared approaches for each type of vulnerability. It is important to note that some vulnerability detection tools (such as Osiris, Oyente, and Securify) fail to identify all three vulnerability types. Consequently, the table includes their results only for the corresponding vulnerabilities, and we do not report an average value for these tools.

The initial comparison involves Clear and seven rule-based techniques: sFuzz, Smartcheck, Osiris, Oyente, Mythril, Securify, and Slither. The performance of these methods is presented in lines 1 to 7 of the table. We observe that Clear exhibits a substantial performance improvement over existing rule-based vulnerability detection tools across all three vulnerability types. Specifically, compared with Slither, which is the state-of-the-art tool for RE and TD detection, Clear achieves an F1-score of 96.62% and 96.31% in RE and TD, respectively, representing a significant improvement of 30.62% and 41.22%. For IO detection, Clear outperforms Mythril, which is the state-of-the-art tool for IO detection, by 35.69% in terms of F1-score, achieving 90.64%.

Subsequently, we compare Clear with five state-of-the-art deep learning-based vulnerability detection methods, including GCN, TMP, AME, SMS, and DMT. The performance results of these methods are presented in lines 8 to 12 of Table 1. The experimental results show the effectiveness of Clear in detecting the three vulnerability types compared to existing deep learning-based approaches. The best-performed DMT achieves average recall, precision, and F1-score of 85.13%, 87.28%, and 86.14% for the three types of vulnerability. Our Clear outperforms the DMT on all three metrics. The precision and recall of Clear achieve 93.64% and 95.44%, respectively, representing a significant improvement of 12.11% and 7.29%
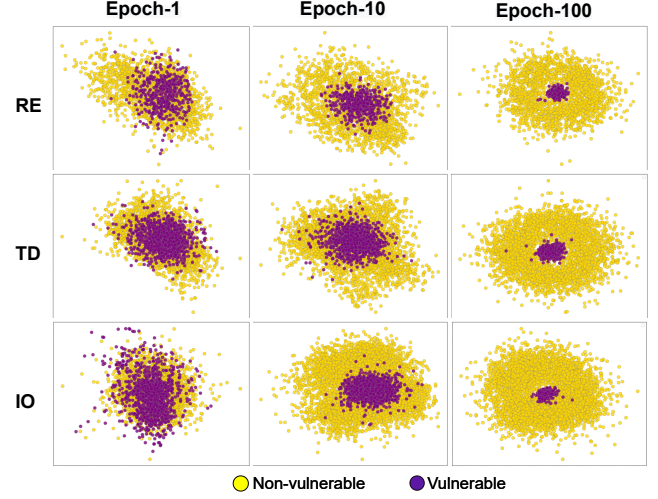

Figure 4: The feature distribution of smart contracts at different epochs during the CL stage.

compared to the average values obtained by DMT, resulting in an overall F1-score of 94.52%.

Finally, in Table 1, line 13 reports the performance of the state-of-the-art general method, LineVul, which uses CodeBERT to detect vulnerabilities. The quantitative results suggest that simple migration of general methods to the SCVD field may not yield satisfactory results. Even the LineVul, which performs the best among the general methods, only achieves an average precision, recall and F1-score of 82.92%, 71.56%, and 76.57% for the three vulnerability detection scenarios, respectively. In comparison, Clear outperforms LineVul across all three metrics, surpassing LineVul by more than 12.93%, 33.37%, and 23.44%, respectively.

> **Answer to RQ1**: The proposed Clear outperforms the state-of-the-art methods across all metrics. On average, Clear achieves an F1-score of 94.52%, showcasing a 9.73% increase in F1-score compared to the existing best-performing method.

5.2 RQ2: Impact of Different Modules

To answer RQ2, we conduct comprehensive ablation tests to examine and understand the impact of different modules on Clear's overall effectiveness. In Section 3.1, we have described that Clear consists of three stages, and therefore, we have specifically designed distinct ablation tests for each of these stages. The results of all ablation tests are presented in Figure 2, in which the metrics **P**, **R**, and **F** represent precision, recall, and F1-score, respectively.

To begin with, for stage 1, we focus our data sampling strategy on two specific types of contract relationships, namely "**V**-**V**" and "**V**-**N**". We selectively mask these relationships in order to evaluate the influence of the labels generated by these two relationships on the overall effectiveness of vulnerability detection. The "Clear-MVV" indicates the masked "**V**-**V**" relationship and "Clear-MVN" indicates

Table 2: The results of the ablation test.

Methods	RE			TD			IO			Average		
	R(%)	P(%)	F(%)	R(%)	P(%)	F(%)	R(%)	P(%)	F(%)	R(%)	P(%)	F(%)
Clear-MVN	75.88	91.86	83.11	90.86	74.68	81.03	82.54	76.47	79.39	83.09	81.00	81.18
Clear-MVV	90.03	81.76	85.44	87.30	83.96	85.60	84.58	81.94	83.24	87.30	82.55	84.76
Clear-RMLM	91.30	90.23	90.77	93.04	90.67	91.85	84.44	88.37	86.36	89.59	89.76	89.66
Clear-RCL	63.25	81.77	71.32	68.48	81.91	71.76	71.62	82.97	76.17	67.78	82.22	73.08
Clear	**96.43**	**96.81**	**96.62**	**98.41**	**94.30**	**96.31**	**91.48**	**89.81**	**90.64**	**95.44**	**93.64**	**94.52**

the masked "**V-N**". As shown in Table 2, Clear-MVN and Clear-MVV achieve 81.18% and 84.76% average F1-score, respectively. In comparison, Clear outperforms both of them and has an F1-score of 94.52%. That is to say, learning only one of the contract relations within the CL stage does not yield satisfactory results. It is only by simultaneously learning both relations that we observe a significant improvement in the performance of SCVD.

Moving on to stage 2, we intentionally remove the MLM module that is integrated into the CL stage. This allows us to analyze the overall effectiveness of the CL stage without the presence of the MLM module. This particular test is referred to as "Clear-RMLM". We observe that the MLM module has a substantial impact on the effectiveness of the CL module. Specifically, when the MLM module is removed, there is an average decrease in precision, recall, and F1-score by 4.15%, 6.13%, and 5.14% respectively. Therefore, we believe that the MLM module can enhance the performance of Clear and is an essential component.

Lastly, for stage 3, we remove the CL stage altogether and directly performed the vulnerability detection stage. This test, known as "Clear-RCL", enables us to evaluate the performance of vulnerability detection in the absence of the CL stage. In comparison to Clear-RCL, we observe a significant improvement in performance for all three types of vulnerability detection tasks with the addition of the CL module. The F-score increased by 35.47% for RE, 34.21% for TD, and 19.00% for IO. This notable improvement can be attributed to the synergistic effects of the CL stage itself and our unique sampling strategy. Specifically, the CL module facilitates the convergence of dispersed vulnerability samples in the feature space, resulting in increased proximity among them. By utilizing our unique sampling strategy, we further reinforce the correlation among samples belonging to the same vulnerability category, thereby promoting their clustering behavior. This process enables the model to more effortlessly identify and discover potential SCVs, leading to a significant improvement in the performance of SCVD.

To substantiate our assertion, we thoroughly examine the derivation process of the sample distribution during the CL stage. In particular, we analyze the evolution of the output of the CL stage (denoted as v in Eq. 7) at each epoch and employ principal component analysis [29] to project each output onto a two-dimensional space. Subsequently, these outputs are visualized as scatter plots and displayed in Figure 4, where the horizontal and vertical axes represent linear combinations of the vectors **v** obtained through PCA. Each point denotes a contract sample, with purple indicating

vulnerability samples and yellow representing non-vulnerability samples. The figure clearly depicts the progression of smart contract sample distribution throughout the CL stage and yields the following finding. First, during the training process of the CL module, the samples of vulnerability contracts exhibit a tendency to cluster together, while being distinctly separated from non-vulnerability samples, indicating a clear distinction between the two groups. Second, this distribution enhances the ability to differentiate and detect SCVs. Clear exhibits a higher proficiency in recognizing this particular cluster and accurately classifying contracts within its proximity as vulnerable. This leads to an improved capability for identifying SCVs.

> **Answer to RQ2**: The two types of relations in contracts are indispensable in the CL stage. The MLM module can enhance the performance of Clear. The CL module facilitates the aggregation of dispersed vulnerability samples in the feature space, leading to a significant performance improvement in the tasks of SCVD.

5.3 RQ3: Effectiveness of the CL Module

To further investigate the contribution of the CL module to other deep learning models in SCVD, we have selected a set of traditional deep learning models, including RNN, LSTM, and GRU, to replace the Transformer model used in Clear. These models, collectively referred to as "CL-Mode," have been specifically chosen because Clear is designed to process code sequences directly, while the deep learning-based methods in the baselines are constructed on graph structures. Therefore, our Clear is incompatible with these graph-based methods. Additionally, we present the performance results of traditional deep learning models to facilitate comparisons and analysis.

The statistical results of these experiments are presented in Table 3 and provide the following observations. All models exhibit notable enhancements in performance when embedded with the CL module. In terms of F1-score, CL-RNN, CL-LSTM, and CL-GRU improved 44.71% (from 48.25% to 69.83%), 40.52% (from 52.88% to 74.30%) and 40.54% (from 55.11% to 77.46%) over the original model, respectively, on the average value of three vulnerabilities.

Table 3: Results of RQ3.

Methods	RE			TD			IO			Average		
	R(%)	P(%)	F(%)	R(%)	P(%)	F(%)	R(%)	P(%)	F(%)	R(%)	P(%)	F(%)
RNN	33.60	37.78	35.56	46.40	69.05	55.50	49.46	58.72	53.70	43.15	55.18	48.25
CL-RNN	64.29	59.34	61.71	76.19	69.06	72.45	82.08	69.60	75.33	74.19(↑ 71.94%)	66.00 (↑ 19.61%)	69.83 (↑ 50.94%)
LSTM	35.71	42.66	38.87	61.11	63.64	62.35	55.56	59.39	57.41	50.79	55.23	52.88
CL-LSTM	74.31	53.26	62.05	83.33	75.00	78.95	86.64	77.67	81.91	81.43(↑ 60.33%)	68.64(↑ 24.28%)	74.30 (↑ 40.51%)
GRU	50.20	35.88	41.85	61.11	63.64	62.35	67.38	55.95	61.14	59.56	51.82	55.11
CL-GRU	80.32	59.69	68.48	81.60	78.46	80.00	89.61	78.86	83.89	83.84 (↑ 40.77%)	72.34 (↑ 39.60%)	77.46(↑ 40.56%)

> **Answer to RQ3**: The empirical evidences suggest the potential of combining traditional deep learning models with the CL module for SCVD, which facilitates the model in acquiring fine-grained feature information and enhancing the performance of SCVD.

6 RELATED WORK

6.1 Smart Contract Vulnerability Detection

SCVD is an important research problem in blockchain security, and numerous scholarly works have been dedicated to exploring it. The initial approaches to detecting SCVs involved static analysis and dynamic execution techniques based on some predefined rules or patterns. For example, Oyente [21] was one of the early SCV detection methods that utilized symbolic execution. It focused on detecting vulnerabilities by analyzing the contract's control flow graph based on symbolic execution. Securify [34] examined the contract's dependency graph and extracted detailed semantic information from the code to identify compliance and security vulnerabilities. Mythril [23] was a static analysis tool that employed concept analysis, taint analysis, and control flow verification to detect common SCVs. TeEther [16] analyzed the contract bytecode and searched for critical execution paths to identify SCVs. Slither [6] was a static analysis framework that converted smart contract source code into an intermediate representation called SlithIR. It utilized this representation to detect SCVs. Osiris [33] combined symbolic execution and taint analysis techniques to detect integer errors in smart contracts. SmartCheck [32], another static program analysis tool, converted Solidity source code into XML and checked for vulnerabilities based on predefined XPath patterns. sFuzz [24] employed a branch distance-driven fuzzing technique to identify vulnerabilities. SMARTIAN [4] was a fuzzier, which utilized lightweight dynamic data-flow analysis to guide fuzzing by collecting feedback based on data flow.

With the advancement of deep learning, there has been a rise in research approaches that harness automated deep learning methods for smart contract vulnerability detection. For example, SaferSC [31] was the first vulnerability detection method to utilize deep learning. It employed a Long Short-Term Memory (LSTM) network to construct a sequence model of Ethereum opcode, providing a comprehensive representation to detect vulnerabilities. More recent deep learning research in this field emphasizes the use of graph structures. DR-GCN [42] transformed smart contract source code into a contract graph with high semantic representation and employed a Graph Convolutional Neural Network (GCN) to construct a vulnerability detection model. TMP [42] extended the approach of DR-GCN by converting critical functions and variables into core nodes with rich semantic information within the contract graph. It also incorporated temporal information on edges. CGE [19] built upon TMP by further incorporating expert mode information, integrating the contract graph information with expert knowledge. AME [18] aimed to combine deep learning and expert mode in an interpretable manner, building upon the CGE approach. DMT [26] proposed a single-modality student network and a cross-modality mutual learning framework to enhance smart contract vulnerability detection on bytecode. However, it is worth noting that all of the methods mentioned above primarily focus on detecting vulnerabilities by learning the semantic knowledge of current input contracts and ignoring the correlation between contracts. Indeed, the inter-contract correlation plays a critical role in understanding the overall security of smart contract ecosystems. Our method successfully improves the performance of SCVD by incorporating correlation information.

6.2 General Vulnerability Detection

Moreover, we also investigate some traditional vulnerability detection techniques that are used to detect vulnerabilities in JAVA, C, and C++ programming languages. The Devign model, proposed by Zhou et al. [40], is a generalized graph neural network-based approach for detecting program vulnerabilities. Its effectiveness was validated through experiments conducted on four different large-scale open-source C projects. Li et al. [17] introduced IVDetect, an interpretable vulnerability detector that leverages deep learning techniques to model program dependency graphs for the purpose of detecting vulnerabilities. Fu et al. [8] proposed LineVul, a Transformer-based approach for detecting vulnerabilities of the C/C++ program at the line level. By utilizing pre-trained models to learn fine-grained code semantic information, LineVul has proven to be the most effective and highest-performing method available. The aforementioned methods, however, fail to yield satisfactory

outcomes in the domain of smart contract vulnerability detection, even with LineVul being considered as the most effective approach, there exists a discernible performance disparity when compared to our method.

6.3 Contrastive Learning

The CL was initially developed as an unsupervised learning technique, aiming to learn representations by uncovering underlying similarities and dissimilarities among samples.

In the field of computer vision, several unsupervised learning methods have proposed CL techniques. Notably, InstDisc [37] was an unsupervised method widely used in computer vision for learning the representation of data. Its goal was to map samples from the same class to similar representation spaces, while samples from different classes were mapped to different representation spaces. InvaSpread [39] proposed an end-to-end learning mechanism that could perform positive and negative sample comparisons within the same mini-batch. MoCo [11] introduced a contrastive learning method based on a dynamic dictionary and dynamic negative sample queue, which improved the quality of feature representation by constructing a large dynamic dictionary to extend the positive sample set. SimCLR [2] was a simple framework for contrast learning representations. SimCLR learned image representation by maximizing the similarity between different views of the same image and achieved significant performance improvements on tasks such as image classification, object detection, and semantic segmentation. SwAV [41] learned image representation by introducing the idea of clustering, assigning samples to different cluster clusters, and achieved impressive results on tasks such as unsupervised image segmentation, object detection, and image classification. SimSiam [3] proposed a simple framework for self-supervised learning to learn the representation of images or features. The core idea was to learn the representation of features by minimizing the Euclidean distance between different views of the same sample using an autoencoder. Its advantages lay in its simplicity and efficiency, without the need to use complex contrast loss functions or negative sample mining strategies.

In the NLP field, ConSERT [27] utilized various data augmentation techniques to construct positive sample pairs, such as cutoff, shuffle, adversarial learning, and dropout. SimCSE [9] used a simple "dropout twice" technique to construct positive sample pairs for CL, achieving a new state-of-the-art performance in unsupervised semantic similarity tasks. ESimCSE [36] later introduced a momentum CL method to construct negative sample pairs. The R-Drop method is similar to SimCSE, applying the "dropout twice" technique to supervised tasks.

Notably, we employ CL for the first time in the SCVD domain and utilize correlation labels to guide the training of the CL model, which play a crucial role in fitting correlation features by the model and effectively enhance the performance of SCVD.

7 THREATS OF VALIDITY

Threats to external validity arise from the datasets and studied vulnerabilities. To minimize the former threat, we utilize the largest publicly available vulnerability dataset that consists of smart contracts labeled as either vulnerable or non-vulnerable. Additionally,

we focus on evaluating the studied vulnerability detection methods on the three most severe and common vulnerabilities, further enhancing the external validity of our research.

Threats to internal validity stem from the implementation of Clear and the compared vulnerability detection methods. To address these threats, we implement Clear using the PyTorch framework and leverage established third-party libraries. Moreover, we utilize the reproducible package of the compared methods, ensuring a fair and standardized comparison.

Threats to construct validity arise from the metrics used to measure the performance of the studied vulnerability detection methods. To mitigate these threats, we employ widely accepted evaluation metrics, such as precision, recall, and F1-score. These metrics provide a comprehensive assessment of the classification performance, ensuring the construct validity of our research. Besides, hardware devices significantly influence the speed of detection as well as threaten structural validity. The threat is addressed by conducting all experiments in this paper on the same device, resulting in traditional detection tools taking approximately 20 to 60 seconds to detect a smart contract, while deep-learning-based detection methods require less than 1 second for the same task.

8 CONCLUSION

With the rapid development of blockchain technology, transactions based on smart contracts have become increasingly frequent and security has become even more critical. However, the SCVs have emerged as one of the top threats to secure transactions. While numerous methods have been successful in mitigating this threat, the discriminative power of existing methods on SCVs still has a lot of room for improvement. Because they fail to explore fine-grained information from vulnerability labels and take into account correlations among contracts. To address these issues, we propose the Clear model, which leverages the CL model to effectively capture inter-contract correlations. By introducing correlation labels, the model can learn fine-grained correlation information. To validate the effectiveness of our Clear, we conduct extensive experiments on a dataset consisting of over 40,000 real-world smart contracts. These contracts are evaluated against state-of-the-art detection methods and compared to Clear for performance. The results demonstrate that our proposed Clear outperforms all detection methods, thereby improving the overall effectiveness of SCVD.

REFERENCES

[1] Sijie Chen, Hanning Mi, Jian Ping, Zheng Yan, Zeyu Shen, Xuezhi Liu, Ning Zhang, Qing Xia, and Chongqing Kang. 2022. A blockchain consensus mechanism that uses Proof of Solution to optimize energy dispatch and trading. *Nature Energy* 7, 6 (2022), 495–502.
[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.
[3] Xinlei Chen and Kaiming He. 2021. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 15750–15758.
[4] Jaeseung Choi, Doyeon Kim, Soomin Kim, Gustavo Grieco, Alex Groce, and Sang Kil Cha. 2021. Smartian: Enhancing smart contract fuzzing with static

and dynamic data-flow analyses. In *36th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 227–239.

[5] Ching-Yao Chuang, Joshua Robinson, Yen-Chen Lin, Antonio Torralba, and Stefanie Jegelka. 2020. Debiased contrastive learning. *Advances in neural information processing systems* 33 (2020), 8765–8775.

[6] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: a static analysis framework for smart contracts. In *IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain*. IEEE, 8–15.

[7] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[8] Michael Fu and Chakkrit Tantithamthavorn. 2022. Linevul: A transformer-based line-level vulnerability prediction. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 608–620.

[9] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821* (2021).

[10] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. 2018. Madmax: Surviving out-of-gas conditions in ethereum smart contracts. *Proceedings of the ACM on Programming Languages* 2 (2018), 1–27.

[11] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 9729–9738.

[12] Andreas Ibing and Alexandra Mai. 2015. A fixed-point algorithm for automated static detection of infinite loops. *IEEE 16th International Symposium on High Assurance Systems Engineering* (2015), 44–51.

[13] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. Zeus: analyzing safety of smart contracts.. In *Ndss*. 1–12.

[14] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Advances in neural information processing systems* 33 (2020), 18661–18673.

[15] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[16] Johannes Krupp and Christian Rossow. 2018. {teEther}: Gnawing at ethereum to automatically exploit smart contracts. In *27th USENIX Security Symposium*. 1317–1333.

[17] Yi Li, Shaohua Wang, and Tien N Nguyen. 2021. Vulnerability detection with fine-grained interpretations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 292–303.

[18] Zhenguang Liu, Peng Qian, Xiang Wang, Lei Zhu, Qinming He, and Shouling Ji. 2021. Smart contract vulnerability detection: from pure neural network to interpretable graph feature and expert pattern fusion. *arXiv preprint arXiv:2106.09282* (2021).

[19] Zhenguang Liu, Peng Qian, Xiaoyang Wang, Yuan Zhuang, Lin Qiu, and Xun Wang. 2021. Combining graph neural networks with expert knowledge for smart contract vulnerability detection. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[20] Ilya Loshchilov and Frank Hutter. 2018. Fixing weight decay regularization in adam. (2018).

[21] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 254–269.

[22] Benjamin Mariano, Yanju Chen, Yu Feng, Shuvendu K Lahiri, and Isil Dillig. 2020. Demystifying loops in smart contracts. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 262–274.

[23] Bernhard Mueller. 2017. A framework for bug hunting on the ethereum blockchain. *ConsenSys/mythril* (2017).

[24] Tai D Nguyen, Long H Pham, Jun Sun, Yun Lin, and Quang Tran Minh. 2020. sfuzz: An efficient adaptive fuzzer for solidity smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 778–788.

[25] Purathani Praitheeshan, Lei Pan, Jiangshan Yu, Joseph Liu, and Robin Doss. 2019. Security analysis methods on ethereum smart contract vulnerabilities: a survey. *arXiv preprint arXiv:1908.08605* (2019).

[26] Peng Qian, Zhenguang Liu, Yifang Yin, and Qinming He. 2023. Cross-Modality Mutual Learning for Enhancing Smart Contract Vulnerability Detection on Bytecode. In *Proceedings of the ACM Web Conference*. 2220–2229.

[27] Hefei Qiu, Wei Ding, and Ping Chen. 2021. Contrastive Learning of Sentence Representations. In *Proceedings of the 18th International Conference on Natural Language Processing*.

[28] Meng Ren, Fuchen Ma, Zijing Yin, Ying Fu, Huizhong Li, Wanli Chang, and Yu Jiang. 2021. Making smart contract development more secure and easier. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1360–1370.

[29] Sam Roweis. 1997. EM algorithms for PCA and SPCA. *Advances in neural information processing systems* 10 (1997).

[30] Amritraj Singh, Reza M Parizi, Qi Zhang, Kim-Kwang Raymond Choo, and Ali Dehghantanha. 2020. Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. *Computers & Security* 88 (2020), 101654.

[31] Wesley Joon-Wie Tann, Xing Jie Han, Sourav Sen Gupta, and Yew-Soon Ong. 2018. Towards safer smart contracts: A sequence learning approach to detecting security threats. *arXiv preprint arXiv:1811.06632* (2018).

[32] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. Smartcheck: Static analysis of ethereum smart contracts. In *Proceedings of the 1st international workshop on emerging trends in software engineering for blockchain*. 9–16.

[33] Christof Ferreira Torres, Julian Schütte, and Radu State. 2018. Osiris: Hunting for integer bugs in ethereum smart contracts. In *Proceedings of the 34th annual computer security applications conference*. 664–676.

[34] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. 2018. Securify: Practical security analysis of smart contracts. In *Proceedings of the ACM SIGSAC conference on computer and communications security*. 67–82.

[35] Zhiyuan Wan, Xin Xia, David Lo, Jiachi Chen, Xiapu Luo, and Xiaohu Yang. 2021. Smart contract security: A practitioners' perspective. In *IEEE/ACM 43rd International Conference on Software Engineering*. IEEE, 1410–1422.

[36] Xing Wu, Chaochen Gao, Liangjun Zang, Jizhong Han, Zhongyuan Wang, and Songlin Hu. 2021. Esimcse: Enhanced sample building method for contrastive learning of unsupervised sentence embedding. *arXiv preprint arXiv:2109.04380* (2021).

[37] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. 2018. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3733–3742.

[38] Tete Xiao, Xiaolong Wang, Alexei A Efros, and Trevor Darrell. 2020. What should not be contrastive in contrastive learning. *arXiv preprint arXiv:2008.05659* (2020).

[39] Mang Ye, Xu Zhang, Pong C Yuen, and Shih-Fu Chang. 2019. Unsupervised embedding learning via invariant and spreading instance feature. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 6210–6219.

[40] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems* 32 (2019).

[41] Zhenglin Zhu, Yawang Wang, Xichuan Zhou, Liuqing Yang, Geng Meng, and Ze Zhang. 2020. SWAV: a web-based visualization browser for sliding window analysis. *Scientific Reports* 10, 1 (2020), 149.

[42] Yuan Zhuang, Zhenguang Liu, Peng Qian, Qi Liu, Xiang Wang, and Qinming He. 2021. Smart contract vulnerability detection using graph neural networks. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 3283–3290.