International Journal of Information Security (2025) 24:183
https://doi.org/10.1007/s10207-025-01102-3

REGULAR CONTRIBUTION

A One-class variational autoencoder for smart contract vulnerability detection

Shaowei Guan[1] · **Ngai Fong Law**[1]

Published online: 22 July 2025

Abstract
Smart contracts and blockchain technology have revolutionized our transactions and interactions with digital systems, yet their vulnerabilities can lead to devastating consequences such as financial losses, data breaches, and compromised system integrity. Existing detection methods, including static analysis, dynamic analysis, and machine learning-based approaches, have their limitations, such as requiring large amounts of labeled data or being computationally expensive. To address these limitations, we propose a novel approach that leverages a One-Class Variational Autoencoder (VAE) with CodeBERT for data pre-processing to detect vulnerabilities in smart contracts. Our approach achieved a higher F1 score (88.93%) compared to the baselines evaluated, even when labeled data is limited. This paper contributes to the development of effective and efficient vulnerability detection methods, ultimately enhancing the security and reliability of smart contracts and blockchain-based systems. By demonstrating superior performance in imbalanced data scenarios, our method offers a practical solution for real-world applications in blockchain security.

Keywords Smart Contracts · Blockchain · Vulnerability Detection · Variational Autoencoder · Transformer · Blockchain Security

1 Introduction

Smart contracts, a fundamental component of blockchain technology, have revolutionized how we conduct transactions and interact with digital systems. A smart contract is a self-executing program that automates the enforcement and execution of a specific set of rules or agreements between parties [1]. The significance of smart contracts lies in their ability to provide a secure, transparent, and tamper-proof environment for various industries, such as finance, supply chain management, and healthcare [2, 3]. With the rapid development of Web 3.0, the adoption of smart contracts is expected to continue growing, leading to a profound impact on how we live and work.

However, despite their numerous benefits, smart contracts are not immune to vulnerabilities. Common types of vulnerabilities include reentrancy, unsecured direct calls, and unchecked low-level calls, among others [4]. These vulnerabilities can lead to devastating consequences, such as financial losses, data breaches, and compromised system integrity. For instance, the DAO hack in 2016 resulted in the theft of approximately $50 million worth of Ether due to a reentrancy vulnerability [5]. In October 2023, Platypus Finance, a stablecoin on the Avalanche blockchain, suffered its third hack of the year. Attackers exploited a vulnerability in the smart contracts, extracting approximately $2.2 million and causing significant financial losses [6]. The severity of these consequences is amplified by the fact that smart contracts are often immutable, making it challenging to rectify vulnerabilities once they are deployed. This highlights the importance of pre-emptively detecting vulnerabilities in smart contracts, which is the primary focus of this research.

Existing methods for detecting vulnerabilities in smart contracts include static analysis, dynamic analysis, and machine learning-based approaches [7–9]. While these methods have shown promise, they have their own limitations. For instance, static analysis can be time-consuming and may not detect all types of vulnerabilities, while dynamic analysis can be computationally expensive and may not generalize

✉ Shaowei Guan
spiderman.guan@connect.polyu.hk

Ngai Fong Law
ngai.fong.law@polyu.edu.hk

[1] Department of Electrical and Electronic Engineering, The Hong Kong Polytechnic University, Kowloon, Hong Kong

well to new contracts. On the other hand, machine learning-based approaches often require large amounts of labelled data, which can be challenging to obtain, especially in the context of rapidly evolving smart contracts. This limitation is particularly obvious in real-world scenarios, where smart contracts are often updated rapidly, making it challenging to label them in a timely manner manually. Another significant limitation is the data imbalance issue; safe contracts are typically more abundant and more accessible to find, while contracts with vulnerabilities are comparatively rare, leading to highly imbalanced datasets [10].

To address these challenges, we propose a novel approach that utilizes a One-Class Variational Autoencoder (VAE) structure to detect vulnerabilities in smart contracts. Our method learns exclusively from safe smart contracts, leveraging their abundance to construct a network that captures their characteristics. We employ CodeBERT [11] to encode these smart contracts and extract meaningful features for training the VAE. By training solely on safe contracts, the model learns to reconstruct them accurately. Consequently, when it is faced with a vulnerable smart contract, the model is unlikely to reconstruct it well, as it is given limited exposure to the features of such contracts during training. This approach effectively detects vulnerabilities in smart contracts, even in scenarios with limited and imbalanced labeled data. This study seeks to address the following two research questions:

1. Under conditions of limited and unbalanced data, does the One-Class VAE outperform traditional autoencoders in detecting the vulnerability of smart contracts?
2. Under conditions of limited and unbalanced data, will the One-Class VAE outperform current machine learning methods?

Our approach aims to fill the existing gap in current detection methods by leveraging the strengths of VAE in learning from limited data. Key contributions of our research include introducing a novel deep learning framework with a VAE as an alternative method to traditional detection methods. It effectively handles limited labelled data and excels in imbalanced situations, making it a practical choice for real-world applications.

This paper is organized as follows. Section 2 comprehensively reviews existing works in smart contract vulnerability detection. Section 3 details our data preparation and analysis methods. Section 4 presents the proposed methodology, including the One-Class VAE structure and classification via reconstruction error. Section 5 evaluates our approach and compares it with existing methods. Section 6 discusses our principal findings, limitations and future direction, followed by a conclusion in Section 7.

2 Related work

The smart contract vulnerability detection field has evolved significantly, transitioning from traditional static analysis methods to machine learning (ML) and deep learning (DL) approaches. This shift has greatly enhanced the efficiency and effectiveness of vulnerability detection processes in smart contracts. Given the benefits of ML and DL, considerable research efforts have been directed toward exploring these methodologies. The subsequent sections will provide a comprehensive overview of recent ML and DL approaches, including supervised learning and semi-supervised learning, as well as discuss the application of variational autoencoders (VAEs) in other vulnerability detection domains.

2.1 Supervised learning approaches

Supervised learning techniques have been widely adopted in smart contract vulnerability detection, with various models demonstrating promising results. Liu et al. [12] proposed a machine learning approach using Solidity bytecode for smart contract honeypot detection in the Ethereum network. This method demonstrated the potential of ML in identifying deceptive smart contracts designed to lure unsuspecting users. Subsequently, Hao et al. [13] developed SCScan, a Support Vector Machine (SVM) based scanning system for identifying vulnerabilities in blockchain smart contracts. This approach leveraged machine learning to automate the detection process, improving upon traditional static analysis methods. ContractWard, an automated vulnerability detection model for Ethereum smart contracts, was introduced by Wang et al. [14]. This model utilized a combination of machine learning techniques to identify various types of vulnerabilities, further advancing the field of automated detection.

As the field progressed, more complex supervised deep learning models were employed. These models leverage different architectural strengths tailored to the unique characteristics of smart contracts. For instance, Lou et al. [15] proposed an improved convolutional neural network (CNN) for detecting Ponzi schemes in smart contracts. This approach utilizes CNNs' ability to capture spatial and hierarchical patterns, making them particularly effective for analyzing bytecode or opcode sequences in smart contracts. By extracting meaningful features from these representations, CNNs enhance the model's capability to identify vulnerabilities and fraudulent patterns that may not be evident in raw data.

Qian et al. [16] developed an LSTM-based method for automated reentrancy detection in smart contracts, showcasing the effectiveness of recurrent neural networks in this domain. Reentrancy vulnerabilities often involve recursive or looping function calls, and the sequential nature of LSTMs allows them to model these intricate dependencies in execu-

tion flow effectively. By learning the temporal relationships between operations in the contract, the LSTM model significantly improves detection accuracy for such vulnerabilities.

Graph Neural Networks (GNNs) have also been applied for smart contract vulnerability detection as demonstrated by Zhuang et al. [17]. This approach leveraged the structural information inherent in smart contracts to improve detection accuracy. Smart contracts can be represented as graphs, with nodes representing functions or variables and edges representing interactions or data flow. GNNs excel at capturing relationships and dependencies within these graphs, enabling the model to identify vulnerabilities that emerge from complex structural interactions, such as circular dependencies or misused variables.

More recently, transformer-based models have demonstrated their ability to outperform traditional architectures in code analysis tasks. Tang et al. [18] fine-tuned CodeBERT, a pre-trained model designed for code-related tasks, to enhance vulnerability detection. CodeBERT's transformer architecture allows it to process global context and long-range dependencies in smart contract code, identifying subtle vulnerabilities that may be missed by other methods. Tang et al. also compared CodeBERT with Optimized-LSTM and Optimized-CNN, illustrating the advantages of transformers in understanding both the semantic and syntactic nuances of code. Transformers' ability to attend to the entire input sequence simultaneously enables them to capture complex patterns, such as nested function calls or intricate logic, making them particularly effective for this task.

2.2 Analysis of strengths and limitations of existing methods

Adopting ML and DL methods has significantly accelerated the vulnerability detection process in smart contracts. These approaches offer the ability to automatically learn and identify complex patterns associated with vulnerabilities, potentially uncovering issues that traditional static analysis methods may miss. However, a significant limitation of the supervised learning approaches is their reliance on large, well-labelled datasets [16]. The dynamic nature of smart contracts and the rapid evolution of vulnerability types make it challenging to maintain up-to-date and comprehensive labelled datasets [18]. This limitation highlights the need for approaches to effectively utilize limited labelled data while leveraging the availability of abundant unlabelled smart contracts.

To address these challenges, semi-supervised learning approaches have gained traction due to their ability to leverage both labelled and unlabelled data. Jiang et al. [19] introduced VDDL, a deep learning-based vulnerability detection model for smart contracts that utilizes semi-supervised learning techniques to address the constraints of limited labelled data in the domain of smart contract vulnerability detection. Similarly, Sun et al. [20] proposed ASSBert, an active and semi-supervised BERT model for smart contract vulnerability detection. This method combined the power of transformer-based models with semi-supervised learning techniques to improve detection accuracy while reducing the reliance on large amounts of labelled data. These advancements highlight the need for approaches that effectively utilize limited labelled data, leveraging the abundance of unlabelled smart contracts, thereby enhancing the overall detection process.

2.3 The use of VAE in other vulnerability detection domains

Variational Autoencoder (VAE) has shown promise in various anomaly detection domains. For instance, Dong et al. [21] proposed a network abnormal traffic detection algorithm based on VAE, achieving effective and accurate results in identifying new attacks and outperforming other classification schemes. Additionally, Khalid et al. [22] demonstrated the effectiveness of a one-class VAE for detecting fake faces in image datasets. This application showcases the potential of VAE in identifying anomalies or vulnerabilities in high-dimensional data. Dong and Kotenko [23] also proposed to use convolutional VAEs for detecting intrusive activities in Internet of Thing (IoT) settings, thereby strengthening IoT security. While VAE has been predominantly applied to image reconstruction tasks, their application to natural language processing (NLP) tasks, such as smart contract vulnerability detection, remains unexplored and presents an interesting avenue for research. The ability of VAE to learn compact representations of complex data structures makes them valuable for capturing the nuanced patterns and structures present in smart contract code. In light of these considerations, our research aims to explore applying semi-supervised learning techniques, specifically utilizing a VAE-based approach, to address the challenges of limited labelled data and the complex nature of smart contracts in vulnerability detection.

3 Data preparation and analysis

3.1 Data collection

A comprehensive dataset with variability is essential for training and evaluating network performance, comprising both safe and vulnerable smart contracts. In our study, we have incorporated four datasets sourced from existing research and widely used in smart contract vulnerability studies. These datasets exhibit diverse types of vulnerabilities and varying ratios of safe and vulnerable contracts. By integrating

Table 1 Collected Dataset Summary

Dataset	Vulnerable Contracts	Safe Contracts	Total
Messi-Q	2,859	583	3,442
SaferSC	215	1,479	1,694
SolidiFI	343	0	343
Slither	0	2,244	2,244
Total	**3,417**	**4,306**	**7,723**

these datasets into our study, we aim to enhance our model's robustness and generalizability.

Specifically, our dataset comprises:

1. **Messi-Q Smart Contract Dataset** [24, 25]: This dataset is composed of both vulnerable smart contracts and safe ones. The vulnerable contracts are categorized into eight types of vulnerabilities, namely: reentrancy, timestamp dependency, block number dependency, dangerous delegatecall, Ether frozen, unchecked external call, integer overflow/underflow, and dangerous Ether strict equality. We successfully obtained 2,859 vulnerable smart contracts and 583 safe smart contracts from this dataset.
2. **SaferSC Smart Contract Dataset** [26]: An additional 1,694 smart contracts are sourced from this dataset, including 215 vulnerable and 1,479 safe smart contracts. The vulnerable contracts contain three types of vulnerabilities: greedy, prodigal, and suicidal.
3. **SolidiFI Benchmark Dataset** [27]: This dataset consists of buggy contracts injected by 9,369 bugs from 7 different bug types, including reentrancy, timestamp dependency, unhandled exceptions, unchecked send, Transaction Order Dependence (TOD), integer overflow/underflow, and use of `tx.origin`. We extracted 343 non-duplicated contracts from this dataset.
4. **Slither Audited Smart Contracts Dataset** [28]: This dataset contains the Solidity smart contracts verified on `Etherscan.io`, along with a classification of their vulnerabilities according to the Slither static analysis framework. Through this dataset, we have extracted 2,244 safe smart contracts.

Table 1 shows the distribution of the datasets, which includes a total of 7,723 smart contracts. The source code of these smart contracts serves as the primary unit of analysis. A subset of our dataset initially consisted of operation code or contract addresses only. To address this limitation, we employed Etherscan [29], an open platform that provides a transparent and verifiable record of transaction histories and smart contract deployments. By leveraging Etherscan's API, we were able to extract the corresponding source codes for these contracts.

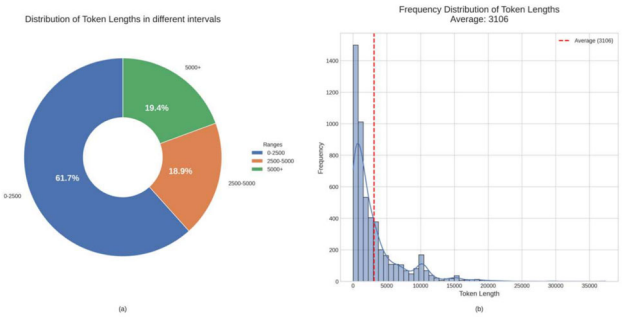

Fig. 1 The analysis of the token length of smart contracts. (a) distribution of token lengths in different intervals and (b) frequency distribution of token lengths

3.2 Data preprocessing

Data Cleaning During the data cleaning phase, we processed the smart contract code to ensure consistency for the CodeBERT model. Specifically, indentation was not preserved, as CodeBERT's tokenization process is primarily based on the sequence of tokens rather than whitespace formatting. Any instance of more than one consecutive space was merged into a single space to standardize the input. Importantly, comments within the smart contract code were preserved during this stage, as they can sometimes provide valuable contextual information about the contract's logic and potential vulnerabilities. However, combining data from different sources can introduce noise into the dataset. To mitigate this issue, we employed a series of data cleaning steps to reduce the noise and ensure data consistency. In particular, we removed unnecessary symbols and characters from the code, including newline characters (`\r \n`, `\n\n`, `\n`), comment delimiters (`//`), and some other common punctuation marks. This step enabled us to focus on the essential tokens and words that convey meaningful information. After removing all these symbols, Figure 1 shows the distribution of the tokens' length of the smart contracts in the dataset. The average length is 3,106 tokens. 61.7% of smart contracts have a token length between 0 and 2,500, 18.9% of contracts with token lengths between 2,500 and 5,000, and those larger than 5,000 tokens constitute 19.4%.

Data splitting The length of smart contracts varied significantly, ranging from hundreds to thousands of tokens (as shown in Figure 1). However, the input for further analysis necessitated uniform input length. For example, the CodeBERT encoder [11], which we employed in our subsequent analysis, has a maximum input length restriction of 512 tokens. To overcome this limitation, we developed a data splitting strategy to divide each smart contract into chunks.

We began by tokenizing each contract using the CodeBERT tokenizer [11]. Subsequently, we split the resulting tokens into chunks, each containing 512 tokens, with a 50% overlap between consecutive chunks. All chunks from one

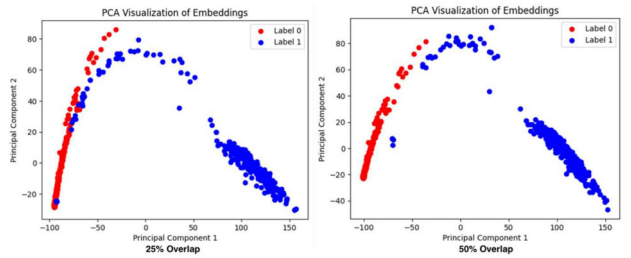

Fig. 2 The PCA Visualization of different overlap strategies. Label 0 and Label 1 represent respectively the safe and the vulnerable contracts

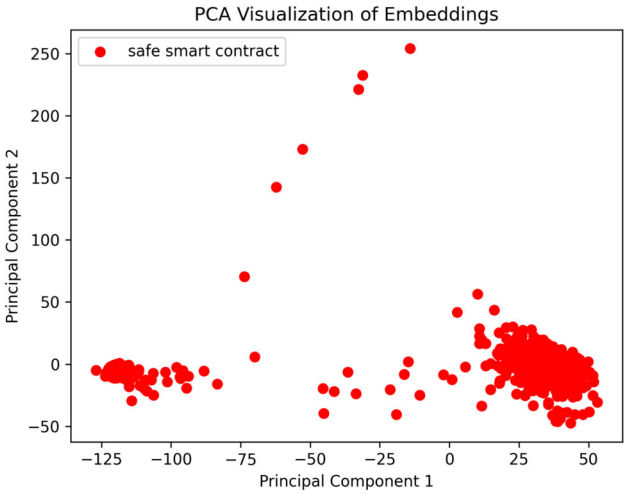

Fig. 3 The PCA Visualization of the safe contracts with token length larger than 5,000

smart contract are assigned the same label. This overlap strategy was designed to capture the contextual relationships between adjacent chunks. To maintain uniformity, chunks shorter than 512 tokens are padded with zeros. To mitigate the impact of extremely long contracts, we decided to include only contacts whose token length is smaller than 5,000 (80.6% of the whole dataset), ensuring that our dataset remained diverse and representative.

The key to our data splitting strategy lies in the overlap mechanism, which allows us to capture the nuanced relationships between chunks. By doing so, we avoid treating each chunk as a separate, independent contract, and instead, characterize the intricate connections within a contract. To determine the optimal overlap ratio, we compared two strategies: 25% and 50% overlap. We applied Principal Component Analysis (PCA) to examine the encoded chunks generated by CodeBERT encoder, separately for each overlap ratio. We randomly selected 300 chunks of safe smart contracts and 300 chunks of vulnerable ones from the SaferSC Smart Contract Dataset [26] for this comparison. Figure 2 illustrates the plot of the first two principal components for both overlap strategies. The left panel represents the 25% overlap strategy, while the right panel represents the 50% overlap strategy. As evident from Figure 2, the 50% overlap strategy exhibits a more distinct clustering pattern and outperforms the 25% overlap strategy. Indeed, the 50% overlap strategy is more effective at capturing the intrinsic characteristics and latent structures of smart contracts than the 25% one due to the higher degree of contextual continuity between adjacent chunks.

To address the potential impact of excessively long contracts on model training, we conducted an analysis of safe contracts exceeding 5,000 tokens in length. These contracts were encoded using CodeBERT with a 50% overlap, and PCA was applied for visualization. As shown in Figure 3, the resulting PCA plot revealed that these longer safe contracts did not cluster effectively, instead exhibiting two distinct clusters. Including these contracts could negatively influence the training performance of our model. Therefore, we decided to retain only contracts with a token length of less than 5,000.

Data encoding In our research, we leveraged the capabilities of CodeBERT [11], a transformer-based pre-trained language model, to encode the smart contracts and extract meaningful features. CodeBERT is specifically designed to handle various programming languages such as Python, Java, JavaScript, PHP, Ruby, and Go. Its architecture is well-suited to capture the syntactic and structural patterns inherent in these languages. It receives an input sequence with a maximum length of 512 tokens and encodes it into an output with a shape of 512×768 [11]. This transformer-based approach not only enhances feature extraction but also aligns with findings by Liew and Law [30], who emphasize the effectiveness of such models in detecting complex patterns across different contexts.

While smart contracts are written in Solidity, they share similarities with other programming languages regarding syntax and structure. By utilizing CodeBERT, we can tap into its ability to focus on the unique features of Solidity language and extract relevant information effectively. This approach enables us to capture the nuanced characteristics inherent in smart contracts, which may not be adequately represented by other transformer-based models like BERT [31] or DistilBERT [32].

To evaluate the efficacy of CodeBERT in encoding smart contracts, we conducted a comparative analysis of its embeddings with those generated by DistilBERT. For this experiment, we randomly selected a subset of 200 chunks of safe smart contracts and 200 chunks of vulnerable smart contracts from the SaferSC Smart Contract Dataset [26]. We applied PCA to the embeddings generated by both CodeBERT and DistilBERT to visualize and compare the resulting feature spaces. Figure 4 shows the first two principal components of the CodeBERT and DistilBERT embeddings. The CodeBERT embeddings demonstrated a clear separation between the safe and vulnerable smart contract clusters, indicating that the model is capable of capturing the underlying patterns and

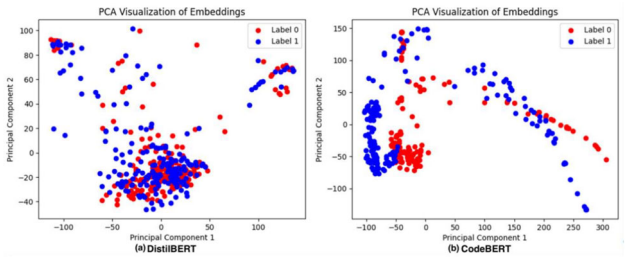

Fig. 4 The PCA visualization of embeddings generated by (a) DistilBERT and (b) CodeBERT. Label 0 and Label 1 represent respectively the safe and the vulnerable contracts

features that distinguish between these two classes. In contrast, the DistilBERT embeddings exhibited a more scattered and overlapping distribution, suggesting that the model struggles to disentangle the relevant features and patterns in the smart contract data.

Data deduplication To mitigate the risk of data leakage due to potential overlap between the different dataset sources, after encoding the smart contract content using CodeBERT, we implemented a deduplication check based on the generated embeddings. This process ensured that no identical contract embeddings were present in both the training and testing datasets, thus confirming the proper separation of our evaluation sets.

4 Methodology

Our proposed methodology addresses two key challenges inherent in smart contract vulnerability detection: the data imbalance problem and the limited availability of labelled data in the real world. We propose an approach utilizing a One-Class Variational Autoencoder (VAE). As illustrated in Figure 5, we used the well-processed safe smart contract to train the VAE, allowing the model to learn the characteristics of these safe contracts. We then evaluate the performance on a group of test data with both safe and vulnerable contracts. Since the trained model learns features of safe contracts, it should reconstruct them accurately, resulting in a small reconstruction error. In contrast, vulnerable contracts will not be reconstructed effectively, leading to a large reconstruction error. By setting a threshold based on the reconstruction error, we can effectively determine whether the test contract is safe or vulnerable. The following subsections discuss details regarding the structure of VAE, training process, and threshold setting.

4.1 Proposed one-class VAE structure

A VAE is a type of generative model that learns to compress and reconstruct input data by optimizing a probabilistic objective function [33]. Figure 6 shows the structure of the proposed VAE. It consists of two main components: an encoder and a decoder. The encoder network is built using a long short term memory network (LSTM) to better capture the sequential relationships of the tokens in the smart contracts [34]. The decoder network attempts to reconstruct the input data. By learning the underlying structure of the safe smart contracts, VAEs can generate data that resembles the safe contracts.

Specifically, the encoder transforms input data x into a latent space z by outputting the parameters of a Gaussian distribution, i.e., the mean μ and log variance $log(\sigma^2)$:

$$q(z|x) = N(z; \mu(x), \sigma^2(x)) \tag{1}$$

From the Gaussian distribution defined by the encoder, we sample latent variables z:

$$z = \mu(x) + \sigma(x) \cdot \epsilon, \epsilon \sim N(0, 1) \tag{2}$$

In this step, we introduce stochastic noise to the latent variables z. Subsequently, the decoder utilizes the latent variable z to reconstruct the original input data x. This process can be mathematically expressed as:

$$p(x|z) = N(x; \widehat{x}(z), \sigma^2) \tag{3}$$

Here, $\widehat{x}(z)$ represents the reconstructed output, which should closely resemble the original input if the model has learned effectively. The training of the VAE is guided by maximizing the Evidence Lower Bound (ELBO). The ELBO is expressed as follows:

$$L(x) = E_{q(z|x)} log p(x|z) - D_{KL}(q(z|x) \parallel p(z)) \tag{4}$$

where $E_{q(z|x)}[\log p(x|z)]$ is the expected log-likelihood term, which encourages the decoder to accurately reconstruct the input data from the latent variable z. $D_{KL}(q(z|x) \parallel p(z))$ is the Kullback-Leibler (KL) divergence that measures the difference between the approximate posterior.

As shown in Figure 6, the encoder begins with an input layer that accepts data in a specified shape of 512×768, which corresponds to the output shape produced by CodeBERT. It then processes the input through three LSTM layers: the first with 512 units, the second with 256 units, and the third with 128 units. This layered approach allows the model to learn complex patterns in the data. Following the LSTM layers, a dense layer with 64 units and a ReLU activation function is added to further refine the representation. A dropout layer with a rate of 0.2 is included to prevent overfitting. The dimension of the latent space is 64 in our model.

The decoder begins with an input layer that receives samples from the latent space, with dimensions of 64. Initially,

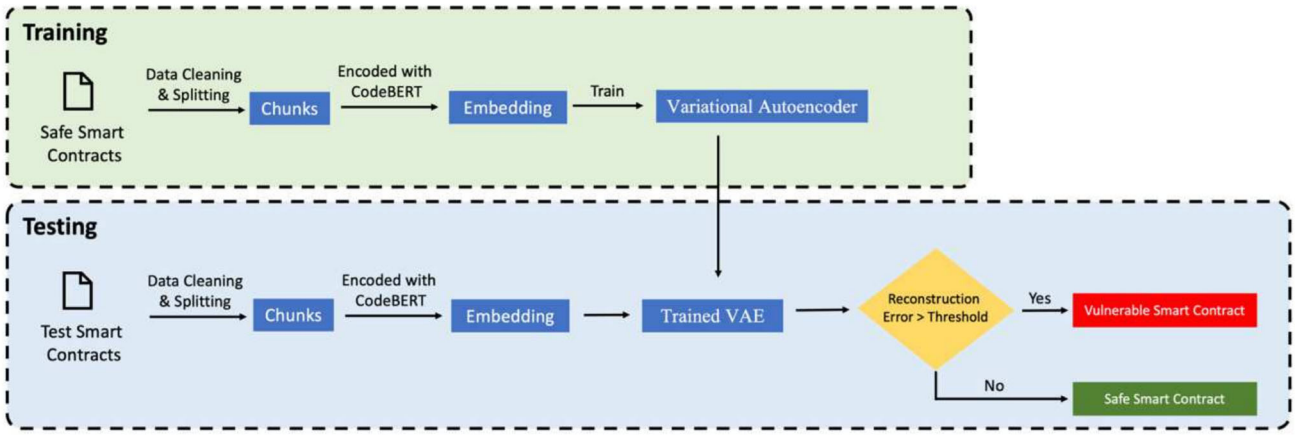

Fig. 5 The workflow of our proposed method

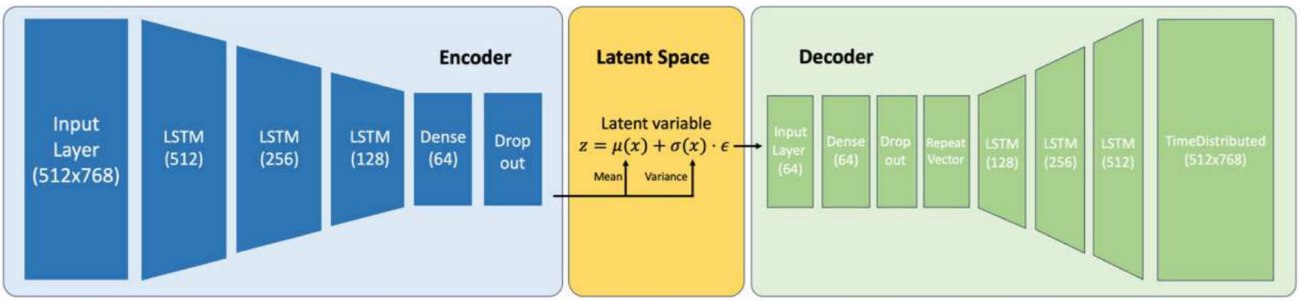

Fig. 6 Our proposed Variational Autoencoder (VAE) Architecture with latent space

a dense layer with 64 units, using a ReLU activation function, processes the latent representation. This is followed by a dropout layer with a rate of 0.2 to help prevent overfitting. Before the LSTM layers, a RepeatVector layer is used to reshape the latent input to match the sequence length of the original input data. The model then employs three LSTM layers, configured in an expanding structure with 128, 256, and 512 units, respectively. This layered approach allows the model to progressively reconstruct the compressed information. The output of the decoder is produced through a final TimeDistributed layer, which applies a dense layer to each time step, reconstructing the input data's original shape (512×768).

Given the real-world scenario where safe smart contracts outnumber vulnerable ones, we design a one-class model that utilizes only safe contracts for training. During training, we use 5,000 chunks of safe smart contracts (label 0) with a batch size of 32 and 15 training epochs to optimize the VAE's parameters.

4.2 Classification via reconstruction error (MAE)

By training the VAE on safe smart contracts only, we expect the model to reconstruct safe smart contracts accurately. However, when presented with a vulnerable smart contract, the VAE is unlikely to reconstruct it well, as it has not learned its features during training. To quantify this reconstruction error, we calculate the Mean Absolute Error (MAE) between the input and output matrices. Each sample has an MAE value, which is calculated by averaging the element-wise differences between the input and output matrices. Mathematically, the MAE is written as,

$$MAE = \frac{1}{n} \sum_{i=1,j=1}^{n} |x_{ij} - \widehat{x}_{ij}| \tag{5}$$

where n is the total number of elements (512×768) in the matrices, x_{ij} denotes the i-th row and j-th column element of the input matrix, and $\widehat{x}_{ij}$ denotes the corresponding element of the output matrix. This average value serves as the reconstruction error for the smart contract.

As mentioned above, our approach is based on the principle that smart contracts with higher reconstruction errors are more likely to be vulnerable. By setting a threshold on the reconstruction error, we can classify contracts as safe or vulnerable. If the reconstruction error exceeds the threshold, the contract is classified as vulnerable; otherwise, it is classified as safe. This approach enables us to detect vulnerabilities in smart contracts by identifying deviations from normal behaviours, as captured by the VAE's reconstruction error.

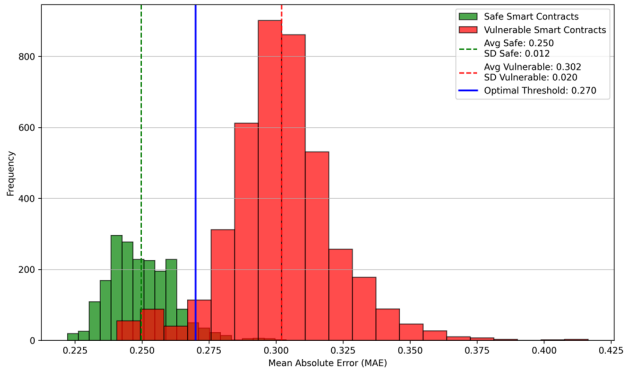

Fig. 7 Distribution of Mean Absolute Error (MAE) between safe and vulnerable smart contracts

4.3 Threshold setting

To determine the decision threshold for classifying contracts as safe or vulnerable, we use a separate dataset of 6,136 chunks, comprising 2,000 safe and 4,136 vulnerable smart contracts, mainly extracted from Messi-Q Smart Contract Dataset [24, 25]. We calculate the MAE for each contract. As shown in Figure 7, the distributions of MAE values differ between safe and vulnerable smart contracts. For safe contracts, the reconstruction errors fall within the range of 0.222 to 0.304 (Mean:0.250, SD:0.012), whereas vulnerable contracts exhibit errors ranging from 0.241 to 0.416 (Mean:0.302, SD:0.020). This discrepancy reflects the model's challenges in accurately capturing the features of vulnerable contracts due to the lack of similar training data. Such distinctions are crucial for effectively identifying and classifying smart contracts based on their vulnerability status.

To obtain the optimal decision threshold, we evaluated the F1 score across various threshold values. The threshold that yielded the highest F1 score was selected as the optimal decision threshold for our VAE. In our model, the best threshold was determined to be 0.270 as shown in Figure 7, achieving an F1 score of 96.28%. This high F1 score underscores the model's effectiveness in accurately classifying smart contracts. Additionally, the threshold of 0.270 corresponds to the mean MAE of safe contracts plus 1.7 standard deviations. Therefore, when employing our method, the threshold can be dynamically adjusted based on the MAE distribution of a user's own safe smart contracts.

5 Evaluation

5.1 Evaluation settings

We utilized three distinct subsets of our dataset throughout the process. First, we trained the VAE using 5,000 chunks of safe smart contracts from the SaferSC Smart Contract Dataset, which were processed with CodeBERT. Next, we determined the threshold for the VAE using another subset of contracts from the Messi-Q Smart Contract Dataset. Finally, our testing dataset comprised 660 chunks of safe smart contracts and 4,200 chunks of vulnerable smart contracts sourced from the SolidiFI Benchmark Dataset and Slither Audited Smart Contracts Dataset. This comprehensive approach allowed us to thoroughly evaluate the model's ability to distinguish between safe and potentially risky code.

To evaluate the performance of our proposed method, we employ a range of metrics, including accuracy, F1 score, recall, and precision. These metrics provide a comprehensive assessment of our model's ability to detect vulnerabilities in smart contracts.

1. Accuracy measures the proportion of correctly classified instances.
2. F1 score is the harmonic mean of precision and recall, providing a balanced measure of both.
3. Recall represents the proportion of true positive instances among all actual positive instances.
4. Precision measures the proportion of true positive instances among all predicted positive instances.

5.2 Comparison with other autoencoders (semi-supervised learning)

There are various ways to construct autoencoders. In our exploration of research question 1, we compare the performance of our VAE with two other autoencoder structures: a dense layer-only autoencoder and an LSTM layer-only autoencoder. The structures of these two autoencoders are shown in Figure 8. The LSTM layer-only autoencoder has the same structure as our VAE, except that the VAE imposes a Gaussian distribution structure on the latent space. The training data for these two autoencoders is the same as that of the VAE. We employ the same method and the same data to determine the optimal threshold for each autoencoder. The resulting thresholds are 0.30 for the dense-only autoencoder and 0.27 for the LSTM-only autoencoder.

Table 2 provides a summary of the performance results of our VAE and the two autoencoders in terms of accuracy, recall, precision, and F1 score. Under the same layer structure, the VAE demonstrated a distinct advantage in effectively classifying safe and vulnerable smart contracts as its F1 score is higher than the other two structures. This comparison demonstrates that the mechanisms of the VAE, specifically its ability to learn a probabilistic latent space, enhance its performance over these traditional autoencoder architectures.

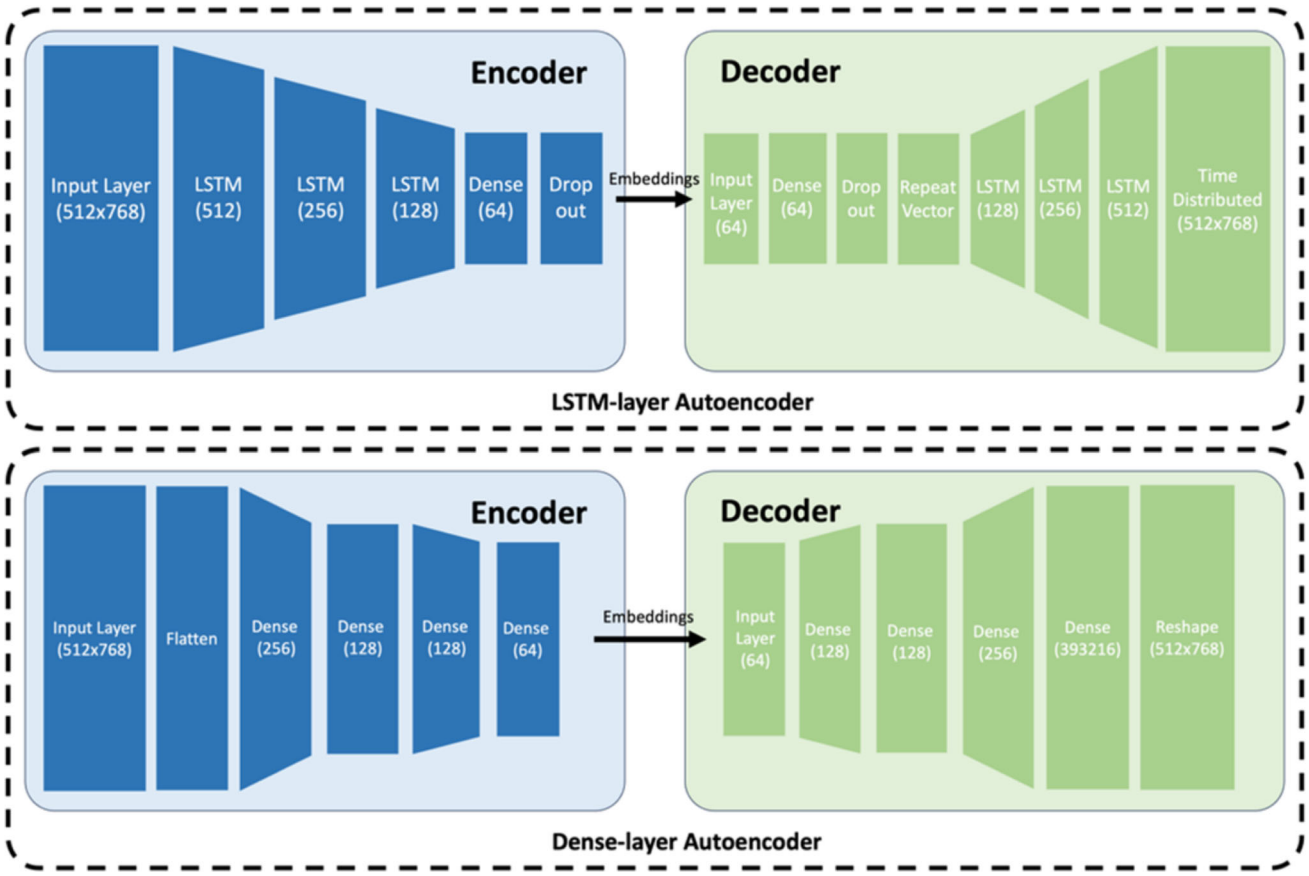

Fig. 8 The structure of the Dense-layer autoencoder and LSTM-layer autoencoder

Table 2 The comparison between the VAE and other autoencoders

Model	Accuracy	Recall	Precision	F1 score
Dense-autoencoder	0.8140	0.8071	**0.9730**	0.8824
LSTM-autoencoder	0.8064	0.8002	0.9705	0.8772
VAE	**0.8193**	**0.8398**	0.9451	**0.8893**

5.3 Comparison with other supervised models

In this study, we compare the performance of our proposed VAE with five supervised learning models commonly used in smart contract vulnerability detection: support vector machine (SVM) [13], random forest (RF) [35], convolutional neural network (CNN) [15], recurrent neural network (RNN) [16], and fine-tuned DistilBERT [36]. These models were selected because they represent a range of traditional and state-of-the-art approaches to smart contract vulnerability detection, each with distinct methodologies that have demonstrated effectiveness in prior research.

The SVM and RF models were included as representative traditional machine learning approaches. SVM has been extensively applied in smart contract vulnerability detection due to its ability to classify vulnerabilities with high precision in relatively small datasets [13]. RF, a widely used ensemble learning algorithm, is known for its robustness and effectiveness in handling class imbalance, which is a critical issue in smart contract datasets [35].

Deep learning methods such as CNNs and RNNs have gained traction for their ability to learn complex patterns from smart contract data. CNNs have been shown to excel in processing structured representations of bytecode, enabling accurate vulnerability detection through hierarchical feature extraction [15]. RNNs, on the other hand, are effective at analyzing sequential data such as execution traces or call graphs, making them particularly suited for detecting temporal vulnerabilities like reentrancy attacks [16].

The fine-tuned DistilBERT model represents a more recent development in the field, leveraging transformer architectures pre-trained on large-scale code datasets. DistilBERT has been shown to achieve state-of-the-art results in vulnerability detection tasks by capturing both semantic and syntactic nuances in smart contract code [36]. Its inclusion ensures that our comparison addresses both traditional machine learning and cutting-edge transformer-based methods.

Table 3 The comparison between the proposed VAE and other supervised learning models

Model	Accuracy	Recall	Precision	F1 score
SVM	0.7811	0.7895	0.9485	0.8617
RF	0.7163	0.6781	**0.9906**	0.8051
CNN	0.8014	0.8264	0.9363	0.8779
RNN	0.7848	0.7974	0.9450	0.8649
Fine-tuned DistilBERT	0.6636	0.7650	0.8322	0.7972
VAE	**0.8193**	**0.8398**	0.9451	**0.8893**

To ensure a fair evaluation, all baseline models were trained on the same labeled dataset used for the VAE, which included 5,000 chunks of safe smart contracts and 50 chunks of vulnerable smart contracts, with an additional 1% of vulnerable smart contracts added to simulate real-world class imbalance. This setup allows us to benchmark the models under identical conditions and assess their robustness in handling imbalanced data.

The comparative results, presented in Table 3, demonstrate that the proposed VAE outperforms all other models in terms of accuracy (81.93%), recall (83.98%), and F1 score (88.93%). This highlights its effectiveness in leveraging safe smart contracts to detect anomalies without requiring explicit vulnerability labels. In contrast, supervised methods such as SVM and RF show reduced performance in handling the imbalanced dataset, while CNN, RNN, and fine-tuned DistilBERT, despite their strengths, rely heavily on the quality and quantity of labeled data.

While our One-Class VAE with CodeBERT achieves a superior F1 score compared to several baseline methods, it exhibits lower precision in some instances as shown in Table 2 and Table 3. This suggests that while our model effectively identifies a large proportion of actual vulnerabilities (high recall), it also flags some safe contracts as potentially vulnerable (lower precision). This could be attributed to the inherent nature of the one-class classification approach, which learns a boundary around the normal (safe) contracts. Contracts that lie outside this boundary are classified as anomalies, and some safe contracts with unusual code patterns might fall into this category. However, the higher F1 score indicates a better overall balance between precision and recall, which is crucial in security applications where missing vulnerabilities (low recall) can be more detrimental than having false positives (low precision).

5.4 Model size, training efficiency, and deployment considerations

Our One-Class VAE model consists of an encoder with 3,624,640 trainable parameters (approximately 13.83 MB) and a decoder with 2,150,720 trainable parameters (approximately 8.20 MB). The model was trained on a single NVIDIA A100 GPU for 30 epochs with a batch size of 32, with the total training time being 1256.67 seconds. This relatively modest size and training time suggest reasonable computational efficiency. For deployment feasibility in large-scale blockchain systems, the inference time for a single smart contract is substantially lower than the training time (approximately 29.34 ms), making real-time or near real-time vulnerability detection a possibility. Further optimization techniques, such as model quantization and efficient inference pipelines, could be explored to enhance deployment on resource-constrained environments. The model's reliance on CodeBERT embeddings, which can be pre-computed, also contributes to faster analysis during deployment.

6 Discussion

6.1 Principal findings

Our primary contribution is the introduction of a novel one-class VAE framework integrated with CodeBERT for smart contract vulnerability detection. While both CodeBERT and VAEs have been employed in software engineering and security domains, our approach offers specific novelty in the context of smart contract vulnerability detection through the synergistic combination of these techniques in a one-class learning framework [18]. Unlike prior CodeBERT-based methods that primarily focus on supervised classification with labeled vulnerable and safe contracts, our One-Class VAE leverages CodeBERT's semantic understanding of code to learn a representation of normal (safe) smart contracts. This allows us to detect unseen vulnerabilities without requiring extensive labeled data for each specific vulnerability type, addressing a key limitation of supervised methods. Compared to existing semi-supervised anomaly detection methods that might use simpler input features, our integration with CodeBERT enables a more nuanced and semantically informed representation of smart contract code, potentially leading to a more accurate detection of subtle and complex vulnerabilities [19, 20]. The novelty lies in the unique application of a One-Class VAE with CodeBERT embeddings

to tackle the smart contract vulnerability detection problem, particularly in scenarios with limited labeled data.

Furthermore, our study introduces a robust data splitting strategy to address the challenge of varying smart contract lengths, which is critical for ensuring uniform input length for deep learning models like CodeBERT. By tokenizing contracts and splitting them into chunks of 512 tokens with a 50% overlap, we preserve contextual relationships between adjacent chunks, enhancing the model's ability to capture the semantic and syntactic intricacies of smart contracts. This overlap strategy, validated through Principal Component Analysis (PCA), demonstrates that a 50% overlap outperforms a 25% overlap in maintaining contextual continuity and clustering patterns, thereby improving the representation of contract complexities. This methodological innovation not only ensures uniformity in input length but also strengthens the model's capacity to detect vulnerabilities by capturing nuanced relationships within the contract code.

6.2 Limitations and future work

Despite the promising results achieved in this study, several limitations should be acknowledged, each of which suggests directions for future research.

First, the dataset used in this study–comprising 7,723 smart contracts for training and testing–presents a potential limitation in terms of size and diversity. While the model demonstrates strong performance on this dataset, expanding it could enhance generalization and robustness. A larger, more diverse dataset would expose the model to a wider range of secure contract patterns and various types of vulnerabilities, enabling it to learn a more comprehensive representation of safe and unsafe behaviors. Future work should focus on collecting additional smart contracts from diverse sources and employing data augmentation techniques to improve generalizability. For example, Guan et al. [37] proposed using generative AI to generate text data, which addressed the problems of data scarcity and imbalance. Moreover, this study focuses on binary classification (safe vs. vulnerable) without distinguishing between specific vulnerability types (e.g., reentrancy, integer overflow) or contract categories (e.g., DeFi, NFTs, staking). Although a binary approach supports rapid initial assessments, vulnerabilities are often domain-specific. Future research should incorporate contract type metadata, where available, to evaluate the model's performance across different categories and explore its ability to detect specific types of vulnerabilities.

Second, our preprocessing pipeline was designed to align with the input requirements of CodeBERT, involving choices such as removing excessive whitespace and retaining comments [11]. However, the effects of alternative preprocessing strategies–such as excluding comments or modifying indentation–were not systematically evaluated. In the future, the impact of these preprocessing decisions on model performance and robustness should be investigated.

Another potiential limitation of our current approach is the chunking strategy. Given the increasing length of modern smart contracts, we adopted an overlapping chunking approach to preserve contextual relationships between adjacent segments. However, this method may fall short in capturing interdependencies between distant functions or contract sections, especially for vulnerabilities that span multiple components. Future research could explore more sophisticated strategies, such as function-aware chunking, to maintain semantic coherence within chunks. Additionally, techniques for aggregating chunk-level outputs–potentially through call graph analysis–could provide a more holistic understanding of vulnerability patterns.

The black-box nature of deep learning models also poses interpretability challenges, highlighting the importance of developing techniques for greater transparency and explainability. One potential direction for future research is to explore ways to identify the specific areas of the reconstruction error matrix that are most indicative of vulnerable smart contracts. Currently, we calculate the average of the MAE of the reconstruction error matrix, whose dimension is 512×768, but it would be beneficial to develop methods that can pinpoint the exact regions of the matrix that are most different between safe and vulnerable contracts. This could involve developing techniques to visualize and interpret the reconstruction error matrix, or to identify the most important features that contribute to the model's decisions.

Additionally, the approach to threshold selection presents a deviation from a strict one-class learning paradigm. While the VAE is trained solely on safe contracts, the optimal threshold is determined using both safe and vulnerable contracts to maximize the F1 score. Although this enhances practical performance, it introduces a potential bias toward known vulnerability types. Notably, our final threshold selection relies on the distribution of safe contracts. Nevertheless, future research could explore stricter one-class thresholding methods, such as unsupervised clustering or domain-informed heuristics based solely on safe contract characteristics.

By addressing these limitations, future research can further enhance the accuracy, robustness, and interpretability of one-class VAE-based methods for smart contract vulnerability detection.

7 Conclusion

In conclusion, this study introduces a novel one-class VAE approach for smart contract vulnerability detection, demonstrating its effectiveness in identifying vulnerabilities with high accuracy, recall, and F1 score. Addressing key research questions, our findings reveal that the one-class VAE outper-

forms traditional autoencoders and other machine learning methods, especially when data is limited and imbalanced. This research holds significant implications for enhancing the security and reliability of smart contracts, offering a promising solution for developing more secure and trustworthy blockchain applications as smart contracts become increasingly prevalent. We hope this work inspires further exploration and contributes to the advancement of vulnerability detection methods in the field.

Author Contributions S.G. conducted the experiments and wrote the manuscript, with N.F.L. providing suggestions and guidance throughout the research.

Funding Open access funding provided by The Hong Kong Polytechnic University.

Data Availability All datasets used are publicly available.

Declarations

Competing interests The authors declare no competing interests.

References

1. Nzuva, S.: Smart contracts implementation, applications, benefits, and limitations. Journal of Information Engineering and Applications **9**(5), 63–75 (2019). https://doi.org/10.7176/JIEA/9-5-07
2. Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: Platforms, applications, and design patterns. In: Financial Cryptography and Data Security. FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta (2017)
3. Christidis, K., Devetsikiotis, M.: Blockchains and smart contracts for the Internet of Things. IEEE Access. **4**, 2292–2303 (2016). https://doi.org/10.1109/access.2016.2566339
4. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts (SoK). In: Principles of Security and Trust: 6th International Conference, POST 2017, Uppsala, Sweden (2017)
5. Siegel, D.: Understanding the DAO attack. CoinDesk. https://www.coindesk.com/learn/understanding-the-dao-attack/ (2016). Accessed 21 Oct. 2024
6. Behnke, R.: Explained: The platypus finance hack (October 2023). Halborn. https://www.halborn.com/blog/post/explained-the-platypus-finance-hack-october-2023 (2023). Accessed 21 Oct. 2024
7. Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Bünzli, F., Vechev, M.: Securify: Practical security analysis of smart contracts. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 67–82 (2018). https://doi.org/10.1145/3243734.3243780
8. Chang, J., Gao, B., Xiao, H., Sun, J., Cai, Y., Yang, Z.: sCompile: Critical path identification and analysis for smart contracts. In: Formal Methods and Software Engineering: 21st International Conference on Formal Engineering Methods, ICFEM 2019, Shenzhen, China (2019)
9. Wang, S., Zhang, C., Su, Z.: Detecting nondeterministic payment bugs in Ethereum smart contracts. Proceedings of ACM on Programming Languages. 3(OOPSLA), 1–29 (2019). https://doi.org/10.1145/3360615
10. Dattaprasad, P., Vijaya, B.: An overview of blockchain technology: Architecture, consensus, and future trends. Int. J. Adv. Res. Sci. Commun. Technol. 293–298 (2023). https://doi.org/10.48175/IJARSCT-8158
11. Feng, Z., Guo, D., Tang, D., et al.: CodeBERT: A pre-trained model for programming and natural languages. In Findings of the Association for Computational Linguistics: EMNLP 2020. 1536–1547 (2020). https://doi.org/10.18653/v1/2020.findings-emnlp.139
12. Liu, C., Liu, H., Cao, Z., Chen, Z., Chen, B., Roscoe, B.: ReGuard: Finding reentrancy bugs in smart contracts. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, 65-68 (2018)
13. Hao, X., Ren, W., Zheng, W., Zhu, T.: SCScan: A SVM-based scanning system for vulnerabilities in blockchain smart contracts. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). IEEE, 1598–1605 (2020). https://doi.org/10.1109/TrustCom50675.2020.00221
14. Wang, W., Song, J., Xu, G., Li, Y., Wang, H., Su, C.: ContractWard: Automated Vulnerability Detection Models for Ethereum Smart Contracts. IEEE Transactions on Network Science and Engineering. **8**(2), 1133–1144 (2021). https://doi.org/10.1109/TNSE.2020.2968505
15. Lou, Y., Zhang, Y., Chen, S.: Ponzi Contracts Detection Based on Improved Convolutional Neural Network. 2020 IEEE International Conference on Services Computing (SCC), Beijing, China. 353-360 (2020). https://doi.org/10.1109/SCC49832.2020.00053.
16. Qian, P., Liu, Z., He, Q., Zimmermann, R., Wang, X.: Towards automated reentrancy detection for smart contracts based on sequential models. IEEE Access. **8**, 19685–19695 (2020). https://doi.org/10.1109/ACCESS.2020.2969429
17. Zhuang, Y., Liu, Z., Qian, P., Liu, Q., Wang, X., He, Q.: Smart contract vulnerability detection using graph neural networks. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI'20). 454, 3283–3290 (2021). https://doi.org/10.24963/ijcai.2020/454
18. Tang, X., Du, Y., Lai, A., Zhang, Z., Shi, L.: Deep learning-based solution for smart contract vulnerabilities detection. Sci. Rep. **13**(1), 20106 (2023). https://doi.org/10.1038/s41598-023-47219-0
19. Jiang, F., Cao, Y., Xiao, J., Yi, H., Lei, G., Liu, M., Deng, S., Wang, H.: VDDL: A deep learning-based vulnerability detection model for smart contracts. In Proceedings of the International Conference on Machine Learning for Cyber Security, Nadi, Fiji. 72–86. (2023)
20. Sun, X., Tu, L., Zhang, J., Cai, J., Li, B., Wang, Y.: ASSBert: Active and semi-supervised bert for smart contract vulnerability detection. Journal of Information Security and Applications. 73, 103423- (2023). https://doi.org/10.1016/j.jisa.2023.103423
21. Dong, S., Su, H., Liu, Y.: A-CAVE: Network abnormal traffic detection algorithm based on variational autoencoder. ICT Express. **9**(5), 896–902 (2023). https://doi.org/10.1016/j.icte.2022.11.006

22. Khalid, H., Woo, S.S.: Oc-fakedect: Classifying deepfakes using one-class variational autoencoder. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 656-657 (2020)
23. Dong, H., Kotenko, I.V.: Convolutional variational autoencoders and resampling techniques with generative adversarial network for enhancing Internet of Thing security. Pattern Recognit Image Anal. **34**(3), 562–569 (2024). https://doi.org/10.1134/S1054661824700366
24. Qian, P., Liu, Z., Yin, Y., He, Q.: Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode. In: Proceedings of the ACM Web Conference 2023, 2220–2229 (2023)
25. Liu, Z., Qian, P., Yang, J., Liu, L., Xu, X., He, Q., Zhang, X.: Rethinking smart contract fuzzing: Fuzzing with invocation ordering and important branch revisiting. IEEE Trans. Inf. Forensics Secur. **18**, 1–1 (2023). https://doi.org/10.1109/TIFS.2023.3237370
26. Tann, W.J.-W., Han, X.J., Gupta, S.S., Ong, Y.-S.: Towards safer smart contracts: A sequence learning approach to detecting security threats. arXiv preprint (2018). https://doi.org/10.48550/arxiv.1811.06632
27. Ghaleb, A., Pattabiraman, K.: 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, 415–427 (2020)
28. Rossini, M.: Slither Audited Smart Contracts Dataset. https://github.com/mwritescode/slither-audited-smart-contracts (2022). Accessed 1 Oct. 2023
29. Ethereum (ETH) blockchain explorer. https://etherscan.io/ (2024). Accessed 1 Oct. 2024
30. Liew, S.R.C., Law, N.F.: BEAM - An algorithm for detecting phishing link. In: 2022 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), IEEE, 598-604 (2022)
31. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics. (2019). https://doi.org/10.18653/v1/N19-1423
32. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint (2019). https://doi.org/10.48550/arxiv.1910.01108
33. Bond-Taylor, S., Leach, A., Long, Y., Willcocks, C.G.: Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models. IEEE Trans. Pattern Anal. Mach. Intell. **44**(11), 7327–7347 (2022). https://doi.org/10.1109/TPAMI.2021.3116668
34. Liew, S., Law, N.F.: Use of subword tokenization for domain generation algorithm classification. Cybersecurity. **6**, 49 (2023). https://doi.org/10.1186/s42400-023-00183-8
35. Zhang, Y., Kang, S., Dai, W., Chen, S., Zhu, J.: Code will speak: Early detection of Ponzi smart contracts on Ethereum. In: 2021 IEEE International Conference on Services Computing (SCC). 55, 301–308 (2021). https://doi.org/10.1109/scc53864.2021.00043
36. Lê, H.B., Lê, Đ.T.,Đoàn, M.T., et al.: Contextual language model and transfer learning for reentrancy vulnerability detection in smart contracts. In: Proceedings of the 12th International Symposium on Information and Communication Technology. 739–745 (2023). https://doi.org/10.1145/3628797.3628945
37. Guan, S., Hui, V., Stiglic, G., Constantino, R.E., Lee, Y.J., Wong, A.: Classifying the Information Needs of Survivors of Domestic Violence in Online Health Communities Using Large Language Models: Prediction Model Development and Evaluation Study. J. Med. Internet Res. **27**, e65397 (2025). https://doi.org/10.2196/65397