Hindawi
IET Software
Volume 2023, Article ID 6631967, 17 pages
https://doi.org/10.1049/2023/6631967

Research Article

VdaBSC: A Novel Vulnerability Detection Approach for Blockchain Smart Contract by Dynamic Analysis

Rexford Nii Ayitey Sosu,[1,2] **Jinfu Chen**,[1] **Edward Kwadwo Boahen**,[1,2] **and Zikang Zhang**[1]

[1]*School of Computer Science and Engineering, Jiangsu University, Zhenjiang, Jiangsu, China*
[2]*Faculty of Computing and Information Systems, Ghana Communication Technology University, Accra, Ghana*

Correspondence should be addressed to Jinfu Chen; jinfuchen@ujs.edu.cn

Received 6 June 2023; Revised 1 December 2023; Accepted 12 December 2023; Published 29 December 2023

Academic Editor: Alessandro Marchetto

Smart contracts have gained immense popularity in recent years as self-executing programs that operate on a blockchain. However, they are not immune to security flaws, which can result in significant financial losses. These flaws can be detected using dynamic analysis methods that extract various aspects from smart contract bytecode. Methods currently used for identifying vulnerabilities in smart contracts mostly rely on static analysis methods that search for predefined vulnerability patterns. However, these patterns often fail to capture complex vulnerabilities, leading to a high rate of false negatives. To overcome this limitation, researchers have explored machine learning-based methods. However, the accurate interpretation of complex logic and structural information in smart contract code remains a challenge. In this study, we present a technique that combines real-time runtime batch normalization and data augmentation for data preprocessing, along with n-grams and one-hot encoding for feature extraction of opcode sequence information from the bytecode. We then combined bidirectional long short-term memory (BiLSTM), convolutional neural network, and the attention mechanism for vulnerability detection and classification. Additionally, our model includes a gated recurrent units memory module that enhances efficiency using historical execution data from the contract. Our results demonstrate that our proposed model effectively identifies smart contract vulnerabilities.

1. Introduction

Blockchain technology has gained significant traction in recent years, leading to the widespread adoption of smart contracts [1]. However, the increasing utilization of these self-executing programs has also exposed their vulnerability to security flaws, potentially resulting in substantial financial losses. This study aims to address the detection of vulnerabilities in smart contracts written in Solidity, the primary programing language for such contracts.

Existing methods for identifying vulnerabilities in smart contracts predominantly rely on static analysis techniques, which search for predefined vulnerability patterns [2–5]. These manual patterns often fall short of capturing complex vulnerabilities, leading to a high rate of false negatives. To mitigate this limitation, machine learning-based approaches have been explored. However, these methods face challenges in accurately interpreting smart contract code's intricate logic and structural information [6].

Dynamic analysis techniques, which analyze the actual execution of a smart contract, offer a promising alternative for identifying potential security issues. This approach addresses the limitations of static analysis by capturing runtime behavior and anomalies [7]. Nevertheless, effective data preprocessing and feature extraction techniques are crucial for successfully applying dynamic analysis.

In this research, we introduce a technique that employs real-time runtime batch normalization (RT-RBN) and data augmentation for data preprocessing. We also utilize n-grams and one-hot encoding for feature extraction. Our proposed model integrates the strengths of bidirectional long short-term memory (BiLSTM) [8], convolutional neural network (CNN) [9], and the attention mechanism [10] for vulnerability detection and classification. BiLSTM effectively captures temporal dynamics, CNN excels in identifying local

features, and the attention mechanism helps to focus on important parts of the input. Additionally, we incorporate a gated recurrent units (GRU) memory module to enhance the computational efficiency [11].

Our main contributions are as follows:

(1) We introduce runtime batch normalization (RBN) and data augmentation techniques to mitigate overfitting and adapt to changing runtime conditions, thereby enhancing vulnerability detection performance.
(2) We employ n-grams and one-hot encoding for feature extraction, capturing essential opcode sequence information to improve vulnerability detection. This approach allows us to represent complex opcode sequences effectively.
(3) Our proposed model integrates BiLSTM, CNN, and the attention mechanism, effectively capturing both local and long-range dependencies within opcode sequences, thereby enhancing the model's ability to understand smart contract code's complex logic and structure.
(4) We introduce a memory module that stores classification output data, reducing the need for feature reselection and improving computational efficiency.

The remainder of the paper is organized as follows: Section 2 reviews related works on smart contract vulnerability detection methods. Section 3 details our approach. Section 4 outlines the experimental settings. Sections 5 and 6 discuss the experimental results and ablation study, respectively. Section 7 expains the threats to validity. Section 8 provides the brief discussion. Section 9 concludes the paper and suggests avenues for the future research.

2. Related Works

The smart contract vulnerability detection field has seen significant advancements, over the past 2 years. This section aims to comprehensively review these developments, categorizing them into deep learning methods, machine learning methods, and dynamic analysis techniques.

2.1. Deep Learning Methods. The employment of deep learning techniques for smart contract vulnerability detection has increased recently. For instance, Li et al. [5, 12–16]] have focused on detecting reentrancy problems. While these methods offer rapid and accurate vulnerability detection, they often lack a comprehensive comparison with the existing techniques. More recent works, such as by Li et al. [17, 18], have started addressing this gap by offering extensive evaluations and comparisons, enriching the field.

2.2. Machine Learning Methods. There have also been notable advancements in machine learning techniques. Random forest and support vector machines have been applied to smart contract vulnerability detection [19–21]. Recent works like [22–24] have expanded the scope by focusing on multiple types of vulnerabilities, offering a more comprehensive vulnerability detection mechanism.

2.3. Dynamic Analysis of Smart Contracts. Dynamic analysis methods are gaining traction due to their ability to capture runtime behavior and anomalies with works like [25–27] making significant contributions in this area. Recent advancements, such as by Kour and Gupta [28, 29], have started to employ machine learning techniques in conjunction with dynamic analysis, offering a more robust vulnerability detection approach.

2.4. Conclusion. The field of smart contract vulnerability detection has seen advancements across deep learning, machine learning, and dynamic analysis methods. Each approach has its merits but also limitations. Our proposed model aims to integrate the strengths of these diverse methodologies. It employs a hybrid of deep learning and machine learning techniques, specifically BiLSTM, CNN, and the attention mechanism, for nuanced feature extraction and classification. Additionally, the model incorporates dynamic taint analysis to capture real-time behavior, thereby providing a more comprehensive vulnerability detection mechanism. A GRU memory module is also included to enhance computational efficiency. The proposed model is further optimized through a carefully selected set of hyperparameters. Our proposed model offers a holistic, efficient, and robust approach, pushing the boundaries of what is currently achievable in smart contract vulnerability detection.

3. Proposed Method

3.1. Overview. This section delineates a comprehensive framework for detecting vulnerabilities in smart contracts through dynamic analysis of Opcode sequences. The proposed method integrates advanced feature selection techniques, machine learning algorithms, and a GRU memory module. The primary contributions of our proposed method are its novel integration of BiLSTM to handle sequence dependencies in the Opcode data, CNN captures local features through its convolutional layers, and the attention mechanism weighs these features according to their relevance, all working in tandem to achieve superior vulnerability detection performance, as well as the incorporation of a GRU memory module for enhanced efficiency by storing classification output data, which is then fed back into the input stage. This eliminates the need for redundant feature reselection, optimizing computational time, and resources.

This section provides an overview of the proposed model with each subsection providing an in-depth discussion of the individual components and methods incorporated in the proposed model, as illustrated in Figure 1.

The smart contracts dataset used in this study was sourced from SmartBugs [30] and verified using Etherscan [31]:

(1) The dataset undergoes RT-RBN and data augmentation.
(2) The RBN and data augmentation stage explores the preprocessed data in real-time using RF and RBN to resolve issues such as imbalanced data and overfitting to enhance feature selection.

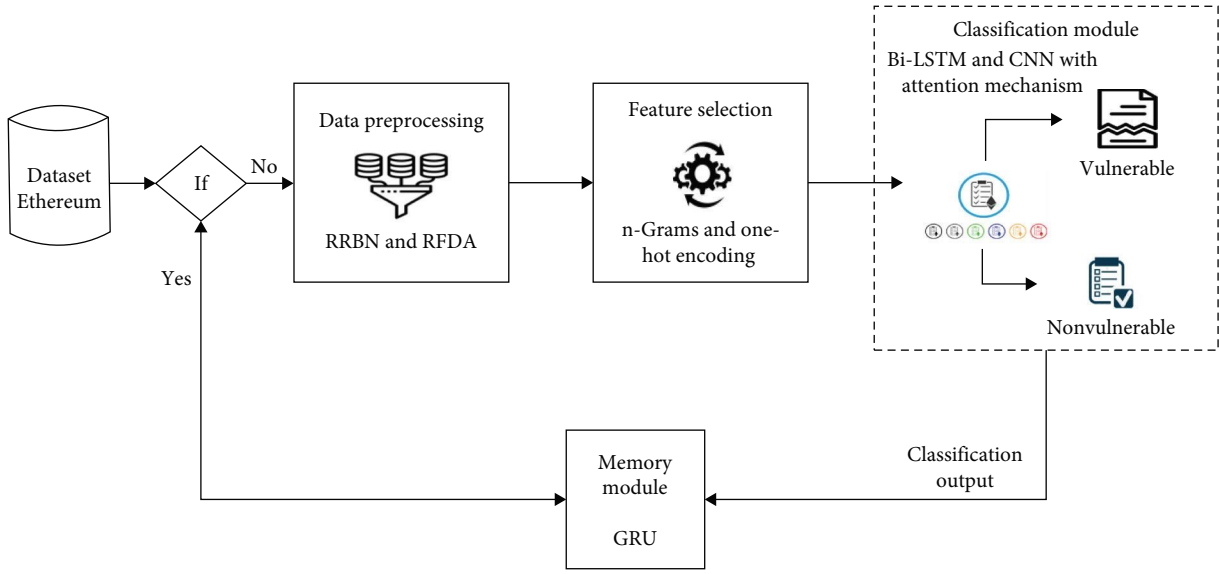

Figure 1: Proposed framework.

(3) Under feature selection, we employed combined techniques such as n-grams and one-hot encoding for feature extraction.
(4) BiLSTM, CNN, and the attention mechanism are integrated and utilized for classification for a better detection output.
(5) The proposed GRU memory module receives data from the classification output. The classification output data transferred to the memory module are then stored and sent to the input module to prevent feature reselection, improving the algorithm's computing time and resources.

Figure 1 provides a schematic representation of the study's workflow. The proposed algorithm, detailed in Algorithm 1, employs dynamic taint analysis techniques for effective vulnerability detection in the smart contracts.

In Algorithm 1, we present a dynamic analysis-based approach for smart contract vulnerability detection based on our proposed model. It incorporates dynamic taint analysis to capture the behavior of smart contracts during runtime and identify potential vulnerabilities based on tainted data. The algorithm follows a step-by-step process that involves preprocessing, feature extraction, model training, and evaluation.

We utilized random flipping (RF) and RT-RBN during preprocessing to improve the dataset's quality. RF introduces variations in the bytecode, while RT-RBN standardizes the execution data. These techniques help address limitations and enhance the overall quality of the contracts' bytecode and execution data.

During the feature extraction phase, the preprocessed bytecode is converted into feature vectors using n-grams or one-hot encoding techniques. The feature vectors also include dynamic features derived from the identified tainted values through dynamic taint analysis. Incorporating dynamic features offers a more comprehensive representation of the smart contract's behavior and possible vulnerabilities.

When training the proposed model, the dataset is split into two phases: training and testing. The proposed model is trained using the training set to identify patterns and features associated with both vulnerable and nonvulnerable contracts.

After training the model, it goes through a phase of evaluation where its ability to detect vulnerabilities is measured using the testing set. This step helps determine how effective the algorithm is in identifying vulnerable contracts.

This subsection introduces the overall architecture and components of the proposed model.

3.2. Data Preprocessing. We utilize the SmartBugs [30] dataset, comprising real-world Ethereum smart contracts, for vulnerability detection. This dataset has been employed in prior research for similar purposes [30, 32]. However, it is imperative to acknowledge the dataset's limitations, such as limited representativeness and selection bias, and to apply appropriate data preprocessing techniques [21, 33].

We apply data preprocessing techniques like RF data augmentation and RT-RBN to mitigate these limitations [34]. The mathematical representation of these techniques is provided in Equations (1) and (2).

Here is the representation of the RF function:

$$f(x) = x \oplus m. \tag{1}$$

The input data, represented by x, is subjected to the bitwise XOR operator denoted by *oplus*. Randomly selected bits in the binary mask m are flipped.

RT-RBN is utilized to standardize the dynamic execution data of smart contracts. This process standardizes the data and reduces the impact of variations caused by various execution environments. As a result, the vulnerability detection

Require: Preprocessed dataset
Ensure: Vulnerability detection model
function DYNAMICTAINTANALYSIS(x)
  Perform dynamic taint analysis to track data flow and identify tainted values in bytecode x
  return Set of tainted values
end function
function FEATUREEXTRACTION(x, T)
  Apply n-grams with one-hot encoding to convert bytecode x into feature vectors
  Incorporate dynamic features derived from tainted values T into the feature vectors
  return feature vectors
end function
Load preprocessed dataset
Feature Extraction Phase:
for each contract c in the dataset **do**
  Extract bytecode b from contract c
  Apply RandomFlipping function using Equation (1) to b: $b' \leftarrow \text{RandomFlipping}(b)$
  Extract execution data e from contract c
  Apply RealTimeBatchNormalization function to e: $e' \leftarrow \text{RealTimeBatchNormalization}(e)$
  Perform dynamic taint analysis on b' and e': $T \leftarrow \text{DynamicTaintAnalysis}(b' \cup e')$
  Extract features f from b' and e' using FeatureExtraction function eqn (3) and eqn (4) with tainted values T
  Replace the original bytecode and execution data in contract c with f
end for
Model Training Phase:
Split the dataset into training and testing sets using Equation (5)
Initialize the BiLSTM-CNN-Attention model
Train the model using the training set using Equation (5)
Model Evaluation Phase:
Evaluate the model using the testing set using Equation (9) for classification
Return classification output

ALGORITHM 1: Proposed Smart Contract Vulnerability Detection.

model can be better generalized across different contracts [35]. Here is the definition of the function for RT-RBN:

$$g(x) = \frac{x - \mu}{\sigma}. \tag{2}$$

The variable x stands for the input data, also known as execution data. μ symbolizes the mean of the data, while σ represents its standard deviation.

Algorithm 2 explains how to use random flipping data augmentation (RFDA) and RT-RBN to improve the quality of smart contract data. RF modifies the bytecode by flipping random bits, while RT-RBN adjusts the execution data by

Require: Dataset (bytecode)
Ensure: Preprocessed dataset
function RANDOMFLIPPING(x)
  Generate a binary mask m with randomly selected flipped bits
  $f(x) \leftarrow x \oplus m$ ▷ Apply bitwise XOR
  return $f(x)$
end function
function REALTIMEBATCHNORMALIZATION(x)
  Calculate the mean μ and standard deviation σ of the execution data
  $g(x) \leftarrow \frac{x-\mu}{\sigma}$ ▷ Apply normalization
  return $g(x)$
end function
Load dataset
Initiate the GRU memory model.
Preprocessing Phase:
for each contract c in the dataset **do**
  Extract bytecode b from contract c
  Apply RandomFlipping function using eqn (1) to b: $b' \leftarrow \text{RandomFlipping}(b)$
  Extract execution data e from contract c
  Apply RealTimeBatchNormalization function using eqn (2) to e: $e' \leftarrow \text{RealTimeBatchNormalization}(e)$
  Replace the original bytecode and execution data in contract c with b' and e'
end for
Return Preprocessed dataset

Algorithm 2: Data Preprocessing Algorithm using RFDA and RRBN.

subtracting the mean and dividing by the standard deviation. These steps help researchers overcome limitations in the dataset and improve smart contract vulnerability detection [3, 4].

First, in the Algorithm 2, we load the dataset and proceed to the preprocessing phase. For every contract in the dataset, the algorithm retrieves the bytecode and execution data. Afterward, it applies the RF and RT-RBN functions to them. Finally, the preprocessed versions replace the original bytecode and execution data.

During the above processes, the RF function uses the XOR operator on input bytecode with a binary mask that RFs bits. The RT-RBN function takes the input execution data, subtracts the mean, and divides it by the standard deviation.

Using data preprocessing techniques, this research effectively mitigates the limitations of the Etherscan dataset, thereby enhancing its quality for model training and evaluation in the context of smart contract vulnerability detection. Therefore, the preprocessed dataset's output becomes more varied, resilient, and appropriate for vulnerability detection.

This subsection details the data preprocessing techniques employed, including RF and RT-RBN.

3.3. Feature Selection. After preprocessing, the dataset is partitioned into training, validation, and testing sets. We employ n-grams and one-hot encoding for feature extraction. These techniques are further elaborated in Equations (3) and (4).

This study employs feature selection techniques that surpass conventional methods' [23, 36] scope, adeptly assimilating local and global features to construct a more resilient model.

The function for extracting n-gram features, denoted as $h(x)$, can be defined as follows:

$$h(x) = x_1, x_2, \ldots, x_n. \tag{3}$$

The variable x represents the input data, and the variable n represents the number of opcodes in each n-gram.

The function for extracting one-hot encoding features, denoted as $h(x)$, is defined as follows:

$$h(x) = [x_1, x_2, \ldots, x_m]. \tag{4}$$

The variable x represents the input data, while m refers to the total number of opcodes in the dataset.

The dataset, named D, has been preprocessed and contains N samples that include input features and corresponding labels. These pairs are shown as: $D = (s_1, y_1), (s_2, y_2), \ldots, (s_N, y_N)$. The input features of the i-th sample are represented by s_i, while its corresponding label is represented by y_i.

Researchers then split the dataset into training, validation, and test sets as follows:

$$\begin{aligned} &D_{Tr}(\text{train}), D_V(\text{val}), D_{Te}(\text{test}) \\ &\quad = S(\text{split})(D, \text{train}_{\text{ratio}}, \text{val}_{\text{ratio}}, \text{test}_{\text{ratio}}), \end{aligned} \tag{5}$$

where $S(\text{split})$ is the function that performs the dataset split, and $\text{train}_{\text{ratio}}$, $\text{val}_{\text{ratio}}$, and $\text{test}_{\text{ratio}}$ represent the desired proportions of the dataset allocated to the training, validation, and test sets, respectively.

The authors prepared the dataset to train deep learning models for detecting smart contract vulnerabilities. We carefully selected a real-world dataset and applied RT-RBN [37] and random flipping approach (RFDA) [38] for preprocessing. The dataset was divided into separate subsets for training, validation, and testing, which ensured an impartial evaluation of the proposed technique. The authors employed feature extraction methods such as n-grams and one-hot encoding to extract meaningful information from the opcode sequence.

We utilized the binary cross-entropy loss function and the Adam optimizer to train the proposed classification model. We train the model using the binary cross-entropy loss function denoted as $\mathscr{L}$. The ground truth label for the i-th training example is y_i, and p_i is the predicted probability of the i-th training example being a vulnerable contract. The learning rate at time step t is η_t, the model parameters at time step t are θ_t, and the mini-batch size is m. To optimize the model, we use the gradient of the loss function concerning the model parameters, denoted as $\nabla_\theta \mathscr{L}$. The total number of training examples is denoted as N.

The model's performance on the test set is evaluated using accuracy, precision, recall, F1-score, and computational time metrics. These metrics were chosen for their relevance in assessing classification models. These measurements helped to determine the proposed model's effectiveness.

This subsection elaborates on the feature selection techniques, including n-grams and one-hot encoding.

3.4. Classification. We integrate BiLSTM, CNN, and the attention mechanism to enhance the classification accuracy. This integration leverages the strengths of each model, providing a more comprehensive analysis of smart contract vulnerabilities. The BiLSTM model effectively captures long-term dependencies in sequential data, while the CNN model excels at learning local patterns in the input. We then utilized the attention mechanism to highlight important features in the input data, improving the model's focus on relevant opcode sequences that contribute to more comprehensive vulnerability detection [34, 39]. The subsequent equations detail the mathematical foundations of the BiLSTM, CNN, and attention mechanisms employed in the proposed model.

The mathematical foundations for these techniques are detailed in Equations (6–8):

$$\begin{aligned} i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) f_t \\ &= \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) o_t \\ &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \widetilde{c}t \\ &= \tanh(W_c x_t + U_c ht - 1 + b_c) c_t \\ &= f_t c_{t-1} + i_t \widetilde{c}_t \, h_t = o_t \tanh(c_t). \end{aligned} \tag{6}$$

In the CNN model, the formula for a 1D convolutional layer is as follows:

$$y_i = \sigma\left(\sum_{j=0}^{k-1} w_j x_{i+j} + b\right). \tag{7}$$

Here is the equation for the attention mechanism:

$$\alpha_i = \frac{\exp(e_i)}{\sum_{j=1}^{n} \exp\left(e_j\right)}, \tag{8}$$

where α_i is the attention weight for feature i, e_i is the relevance score for feature i, and n is the number of features.

In this study, we suggest an integrated model that uses the benefits of BiLSTM, CNN, and the attention mechanism during the classification phase, as Shou et al. [9, 11] have highlighted. Let us denote the input data as X, the output probabilities as P, and the models' weights as W. The equation is expressed as follows:

$$P = \text{Attention}(\text{CNN}(\text{BiLSTM}(X, W_{\text{BiLSTM}}), W_{\text{CNN}}), W_{\text{Attention}}). \tag{9}$$

In Equation (9), $\text{BiLSTM}(X, W_{\text{BiLSTM}})$ represents the BiLSTM model applied to the input data X with weights

W_{BiLSTM}. The output of the BiLSTM model then passes unto the CNN model, $\text{CNN}(\cdot, W_{\text{CNN}})$, using weights W_{CNN}. Finally, the output of the CNN model is passed unto the attention mechanism, $\text{Attention}(\cdot, W_{\text{Attention}})$, with weights $W_{\text{Attention}}$, to obtain the final output probabilities P. Our proposed approach comprehensively analyzes smart contract vulnerabilities, enhancing classification accuracy.

This subsection discusses the integration of BiLSTM, CNN, and the attention mechanism in the classification phase.

3.5. Memory Module. In this research, we integrated a GRU memory module with our proposed model to improve the detection of smart contract vulnerabilities. The GRU is a Recurrent Neural Network (RNN) type that captures long-term dependencies and enhances the representation of sequential data.

The conditional expression for the GRU module is provided below:

If $(U = P)$, then M = reject
Else
If $(U \neq P)$ then M = accept.

In this expression, U represents unprocessed data, P is processed data, and M is the output of GRU. The conditional expression checks whether the unprocessed data U is equal to the processed data P. If they are equal, the GRU output M is set to "reject". Otherwise, if U is not equal to P, the GRU output M is set to "Accept".

To utilize the memory module, researchers input classification results from the output of the proposed classification model to a GRU memory module layer. The classification output data transferred to the memory module is then stored and sent to the input module to prevent feature reselection, improving the algorithm's computational time and resources.

By integrating a GRU memory model, we have enhanced the performance of our proposed model in detecting vulnerabilities in the smart contracts. Our findings show that this integration has resulted in high accuracy and F1-score achievements, which sets this work apart from the existing methods [40].

This subsection discusses the GRU memory module's role in enhancing the proposed model's efficiency.

3.6. Hyperparameter Tuning. The selection of hyperparameters is crucial for the proposed model's performance. Specific choices, such as the learning rate and activation functions, are empirically validated to ensure optimal results. This subsection elucidates the rationale behind selecting specific hyperparameters, including activation functions, learning rates, and batch sizes.

We employed the rectified linear unit (ReLU) activation function in the convolutional layers. The choice is motivated by ReLU's computational efficiency and its ability to mitigate the vanishing gradient problem. We employed the Sigmoid function for the output layer to ensure that the output probabilities lie within the range of [0, 1], making it suitable for the binary classification task.

The learning rate of our proposed model is initially set at 0.001 and dynamically adjusted using the Adam optimizer. The selection of Adam is backed by its adaptive learning rate capabilities, which offer a balanced tradeoff between convergence speed and model accuracy. Empirical evaluations corroborate that this learning rate setting ensures a stable and efficient training process.

We employed a batch size of 32, which offers a compromise between computational efficiency and the stability of the gradient during backpropagation. We observed that smaller batch sizes are computationally expensive and prone to noisy gradients, while larger batch sizes risk overfitting.

Therefore, we employ a RBN technique and incorporate it into our proposed model's architecture. This contributes to faster convergence and mitigates the risk of overfitting, thereby enhancing the proposed model's generalizability.

This subsection discussed the selection and rationale behind the hyperparameters used in the proposed model.

In summary, this study proposes a comprehensive model for detecting vulnerabilities in the smart contracts. The proposed model employs advanced feature selection techniques, integrated machine learning algorithms, and a GRU memory module to improve efficiency and accuracy. The hyperparameters were carefully selected and empirically validated. Our findings indicate that the proposed model offers high accuracy and F1-scores, making it a robust smart contract vulnerability detection approach.

4. Experiment and Evaluation

The proposed research addresses several questions related to dynamic analysis-based smart contract vulnerability detection with interpretability. Researchers aim to comprehensively understand the algorithm's strengths, limitations, and performance characteristics by investigating these research questions in the context of smart contract vulnerability detection. The research questions encompass various aspects of the algorithm, examining the effectiveness of dynamic taint analysis in identifying tainted values, evaluating the advantages and limitations of n-grams and one-hot encoding for feature extraction, assessing the impact of incorporating dynamic features derived from tainted values, investigating the influence of the RF function on bytecode and model performance, examining the contribution of RT-RBN to accuracy and robustness, analyzing the performance metrics of the BiLSTM-CNN-attention-model, exploring and evaluating the detection accuracy for different vulnerability types, and assessing the computational cost and scalability of the algorithm.

4.1. Dataset Description. Our proposed model was meticulously integrated with Remix IDE to automate the detection of vulnerabilities in smart contracts in the SmartBugs [30] dataset. The proposed model is deployed as a web-based API within Remix IDE, synergizing with existing static analysis tools and adding a layer of dynamic analysis. Automated invocation of suspect smart contract functions is executed through the JavaScript VM in Remix IDE, which emulates the Ethereum virtual machine (EVM). The dataset comprises 37,035 smart contracts, of which 8,543 were found to contain vulnerabilities. These vulnerabilities were categorized into 1,100 instances of Integer Underflow, 1,880 instances of Reentrancy, 2,403 instances of transaction ordering dependency

Table 1: Smart contract samples distribution.

Category	Test set	Train set	Total samples
Vulnerable	1,752	6,791	8,543
Nonvulnerable	16,853	3,122	19,975
Reentrancy	1,608	272	1,880
TOD	1,521	882	2,403
Integer overflow	1,696	423	2,119
Integer underflow	680	420	1,100
Unchecked return values	721	320	1,041

(TOD), 1,041 instances of unchecked return values, and 2,119 instances of integer overflow. To create the training and testing sets, 80% of the samples were randomly selected for training, and the remaining 20% were allocated for testing, as illustrated in Table 1.

Preprocessing steps: we applied the preprocessing steps described in detail in Algorithm 2 before the data were utilized for model training. These included RT-RBN to standardize the features and data augmentation techniques like RF to enhance the dataset's robustness.

The dataset used in this study was compared with other commonly used datasets in the field, such as the smart contract weakness classification and test dataset (SWC-CTD) and the Vyper dataset. The dataset from Ethereum's official website and Etherscan was found to be more representative of real-world smart contracts, given its diverse range of vulnerability types.

Uniqueness and representativeness: this research provides a unique dataset in its comprehensiveness and diversity of smart contract vulnerabilities. It includes commonly occurring and less frequent vulnerabilities, thereby providing a more rounded view of the smart contract vulnerability landscape, making the dataset highly representative, and ensuring that the model trained on it is robust and generalizable.

4.2. Experimental Setup. We used a high-performance computing system with specific hardware and software configurations to research smart contract vulnerability detection. Our system had an Intel Core i7 processor and 16 GB of RAM and was operating on macOS Big Sur version 11.7.6. We utilized the R programing language and installed relevant packages such as TensorFlow, Keras, and Caret to support the implementation of our algorithms and models. Our R version was 4.0 or above. This setup provided a reliable and flexible environment for our experiments.

5. Performance Evaluation

5.1. Comparison with Exiting Methods. Our experiment is shown in Table 2, and Figure 2 provides a detailed comparison of different methods to detect vulnerabilities in smart contracts, including our proposed algorithm. We evaluated each technique's effectiveness and efficiency by analyzing precision, recall, accuracy, and F1-measure.

Among the existing tools, Oyente [41] exhibited a precision of 40.9%, recall of 47.6%, accuracy of 60.8%, and an F1-measure of 43.6%, while Maian [41] displayed a precision of

Table 2: The vulnerability detection using diverse methods.

Approach	Precision	Recall	Accuracy	F1-Measure
Oyente	40.9	47.6	60.8	43.6
Maian	63.2	32.5	61.8	48.7
SmartCheck	57.5	52.7	53.8	56.7
Manticore	58.8	42.8	58.5	60.6
ContractGurard	63.7	59.5	80.5	75.3
ContractFuzzer	82.6	63.8	86.6	80.7
Proposed	89.8	93.6	91.5	92.5

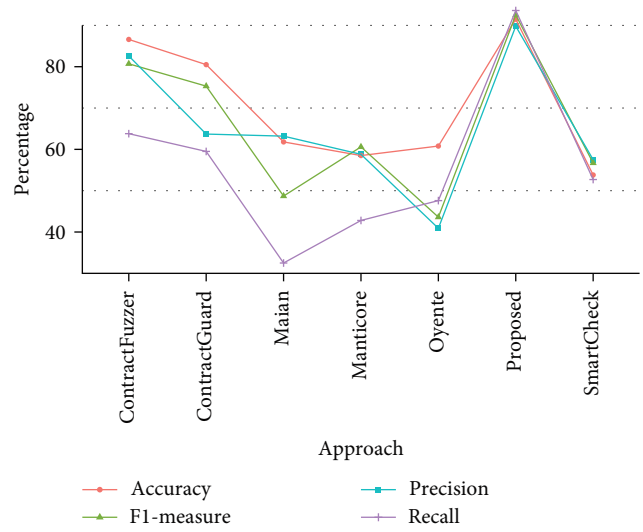

Figure 2: Accuracy, F1-measure, precision, and recall using diverse methods.

63.2%, recall of 32.5%, accuracy of 61.8%, and an F1-measure of 48.7%. SmartCheck demonstrated a precision of 57.5%, recall of 52.7%, accuracy of 53.8%, and an F1-measure of 56.7%. Conversely, Mossberg et al. [42] yielded a precision of 58.8%, recall of 42.8%, accuracy of 58.5%, and an F1-measure of 60.6%. Notably, ContractGuard [43] showed a precision of 63.7%, recall of 59.5%, accuracy of 80.5%, and an F1-measure of 75.3%. Finally, ContractFuzzer [44] displayed an impressive precision of 82.6%, recall of 63.8%, accuracy of 86.6%, and an F1-measure of 80.7%.

However, our proposed algorithm excelled against all existing methods with an impressive precision of 89.8%, recall of 93.6%, accuracy of 91.5%, and an F1-measure of 92.5%. These findings suggest that the newly introduced algorithm is more effective and efficient in detecting vulnerabilities than the other approaches. The high values of precision and recall indicate that the algorithm can accurately detect vulnerabilities while minimizing false positives and false negatives. The overall high accuracy and F1-measure further prove the effectiveness and reliability of the new algorithm in detecting vulnerabilities in smart contracts.

We experimented with different methods of detecting vulnerabilities in smart contracts. Table 3 and Figure 3 presents the accuracy of vulnerability detection for various types and the duration it took for each method to identify them.

Table 3: Comparison of detection accuracy and time of smart contract vulnerability detection methods.

Approach	Reentrancy (%)	DT(s)	Integer overflow (%)	DT (s)	Integer underflow (%)	DT (s)	TOD (%)	DT (s)	URV (%)	DT (s)
Oyente	63.24	130	65.29	128	66.4	125	68.56	120	71.98	130
Maian	60.76	30	62.81	32	63.92	28	66.08	25	69.5	20
SmartCheck	56.37	15	58.42	17	59.53	15	61.69	15	65.11	15
Manticore	73.75	330	75.8	324	76.91	320	79.07	230	82.49	330
ContractGuard	75.68	10	77.73	12	78.84	11	81	10	84.42	10
ContractFuzzer	78.37	3	80.42	2	81.53	5	83.69	2	87.11	3
Proposed	84.51	2	86.56	1	87.67	2	89.83	1	93.25	1

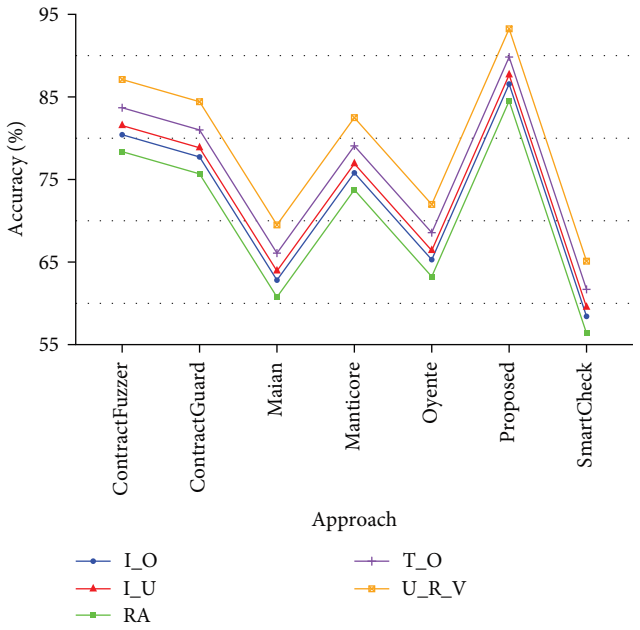

Figure 3: Vulnerability detection accuracy using diverse methods.

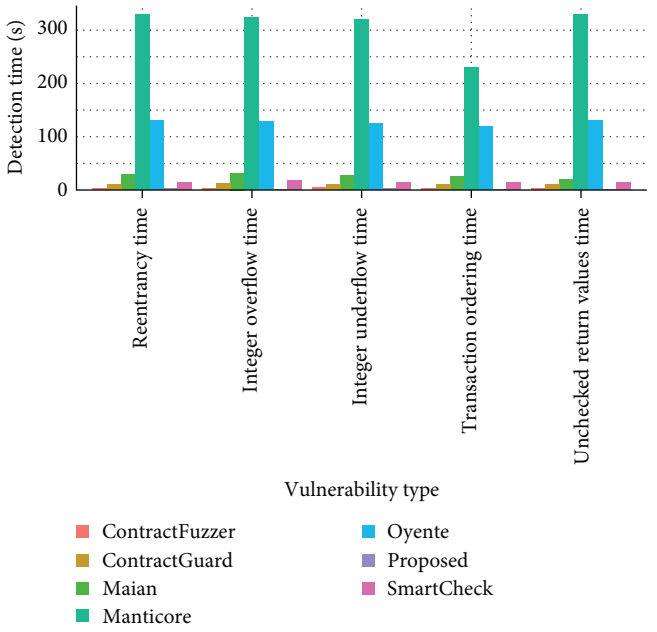

Figure 4: Vulnerability detection time using diverse methods.

Out of all the approaches evaluated, our proposed algorithm exhibited the highest accuracy rate across all vulnerability types. It achieved an accuracy rate of 84.51% for reentrancy vulnerabilities, 86.56% for integer overflow vulnerabilities, 87.67% for integer underflow vulnerabilities, 89.83% for TOD vulnerabilities, and an impressive 93.25% for unchecked return values (URV) vulnerabilities. Based on the results, the proposed algorithm is exceptionally efficient in accurately detecting various vulnerabilities.

Moreover, in Figure 4, our proposed algorithm demonstrated superior efficiency in terms of detection time compared to the other approaches. It achieved detection times of only 2 s for reentrancy, integer overflow, and integer underflow vulnerabilities and 1 s for TOD, and URV vulnerabilities. These short detection times indicate the computational efficiency of the proposed algorithm. Compared to the other approaches, as shown in Table 3 and Figure 3, our proposed algorithm excelled against them in accuracy and detection time for all vulnerability types. For instance, Oyente, Maian, and SmartCheck achieved lower accuracy rates ranging from 56.37% to 63.24%, across the different vulnerability types, while Manticore, ContractGuard, and ContractFuzzer exhibited intermediate accuracy rates ranging from 73.75% to 82.49%. Additionally, the detection times of these approaches varied, with some showing considerably longer detection times compared to our proposed algorithm.

Overall, our experiment results highlight our proposed algorithm's effectiveness and efficiency in detecting vulnerabilities in smart contracts. Its ability to achieve high-accuracy rates and short-detection times makes it a promising solution for identifying vulnerabilities in the smart contracts.

Table 4 and Figure 5 summarize the outcomes of an experiment that assessed the effectiveness of various models in identifying vulnerable and nonvulnerable smart contracts. The experiment evaluated the models based on F1-measure, accuracy, precision, and recall, concentrating on vulnerabilities such as reentrancy, integer overflow, integer underflow, TOD, and URV. The models tested against our proposed model were Oyente [45], Maian [41], SmartCheck [46], ContractGuard [43], and ContractFuzzer [44].

Overall, the models displayed an impressive performance in detecting smart contract vulnerabilities. The F1-measure, which considers precision and recall, ranged from 62.81% to 70.57%, indicating that the models can balance identifying

Table 4: Comparing the existing methods with the proposed model on 60% training data.

Models	Measures	Reentrancy	Integer overflow	Integer underflow	TOD	URV
Oyente	F1-measure	62.81	63.44	63.57	63.66	63.78
Oyente	Accuracy	97.16	97.79	97.92	98.01	98.13
Oyente	Precision	83.51	84.14	84.27	84.36	84.48
Oyente	Recall	70.58	71.21	71.34	71.43	71.55
Maian	F1-measure	63.51	64.14	64.27	64.36	64.48
Maian	Accuracy	96.98	97.61	97.74	97.83	97.95
Maian	Precision	84.41	85.04	85.17	85.26	85.38
Maian	Recall	71.17	71.8	71.93	72.02	72.14
SmartCheck	F1-measure	63.91	64.54	64.67	64.76	64.88
SmartCheck	Accuracy	97.01	97.64	97.77	97.86	97.98
SmartCheck	Precision	83.51	84.14	84.27	84.36	84.48
SmartCheck	Recall	71.41	72.04	72.17	72.26	72.38
ContractGuard	F1-measure	65.24	65.87	66	66.09	66.21
ContractGuard	Accuracy	97.56	98.19	98.32	98.41	98.53
ContractGuard	Precision	84.51	85.14	85.27	85.36	85.48
ContractGuard	Recall	72.62	73.25	73.38	73.47	73.59
ContractFuzzer	F1-measure	65.52	66.15	66.28	66.37	66.49
ContractFuzzer	Accuracy	97.32	97.95	98.08	98.17	98.29
ContractFuzzer	Precision	85.29	85.92	86.05	86.14	86.26
ContractFuzzer	Recall	73.32	73.95	74.08	74.17	74.29
Proposed	F1-measure	70.57	71.2	71.33	71.42	71.54
Proposed	Accuracy	98.46	99.09	99.22	99.31	99.43
Proposed	Precision	88.48	89.11	89.24	89.33	89.45
Proposed	Recall	79.17	79.8	79.93	80.02	80.14

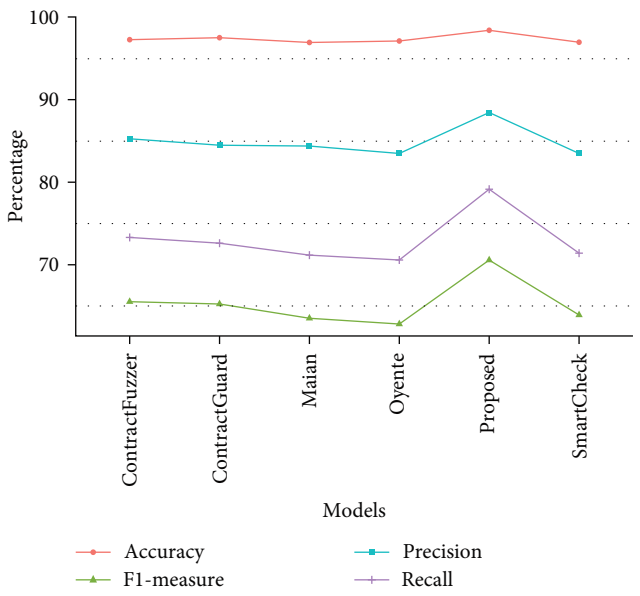

Figure 5: Comparing the existing methods with the proposed model on 60% training data.

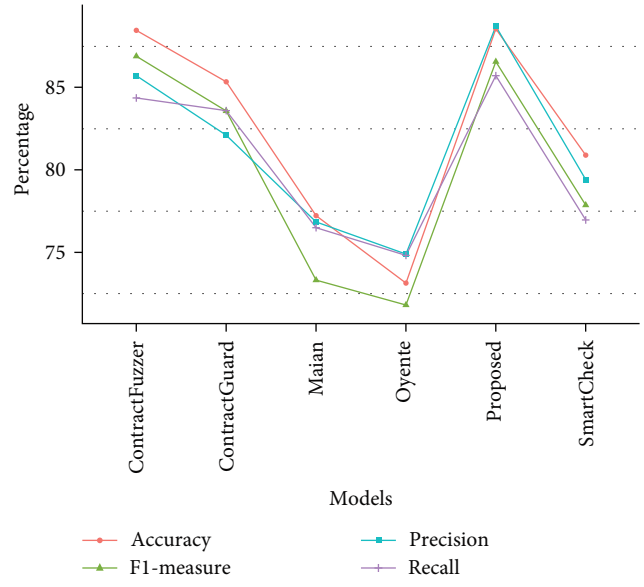

Figure 6: Comparing the existing methods with the proposed model on 75% training data.

true positives while minimizing false positives and false negatives. Accuracy values varied from 96.98% to 98.46%, demonstrating that the models' classification of smart contracts was mostly correct. Precision values ranged from 83.51% to 88.48%, showing that the models could accurately identify true positives. Recall values ranged from 70.58% to 79.17%, indicating that the models could detect actual positives.

Figure 6 and Table 5 provided displays the findings of an experiment that analyzed different models' ability to classify smart contracts according to various vulnerabilities. The evaluation focused on measures such as F1-measure,

Table 5: Comparing the existing methods with the proposed model on 75% training data.

Models	Measures	Reentrancy	Integer overflow	Integer underflow	TOD	URV
Oyente	F1-measure	71.82	72.45	72.58	72.67	72.79
Oyente	Accuracy	73.15	73.78	73.91	74	74.12
Oyente	Precision	74.91	75.54	75.67	75.76	75.88
Oyente	Recall	74.83	75.46	75.59	75.68	75.8
Maian	F1-measure	73.33	73.96	74.09	74.18	74.3
Maian	Accuracy	77.24	77.87	78	78.09	78.21
Maian	Precision	76.86	77.49	77.62	77.71	77.83
Maian	Recall	76.51	77.14	77.27	77.36	77.48
SmartCheck	F1-measure	77.88	78.51	78.64	78.73	78.85
SmartCheck	Accuracy	80.91	81.54	81.67	81.76	81.88
SmartCheck	Precision	79.43	80.06	80.19	80.28	80.4
SmartCheck	Recall	76.98	77.61	77.74	77.83	77.95
ContractGuard	F1-measure	83.59	84.22	84.35	84.44	84.56
ContractGuard	Accuracy	85.36	85.99	86.12	86.21	86.33
ContractGuard	Precision	82.12	82.75	82.88	82.97	83.09
ContractGuard	Recall	83.61	84.24	84.37	84.46	84.58
ContractFuzzer	F1-measure	86.91	87.54	87.67	87.76	87.88
ContractFuzzer	Accuracy	88.48	89.11	89.24	89.33	89.45
ContractFuzzer	Precision	85.73	86.36	86.49	86.58	86.7
ContractFuzzer	Recall	84.38	85.01	85.14	85.23	85.35
Proposed	F1-measure	86.58	87.21	87.34	87.43	87.55
Proposed	Accuracy	88.57	89.2	89.33	89.42	89.54
Proposed	Precision	88.73	89.36	89.49	89.58	89.7
Proposed	Recall	85.73	86.36	86.49	86.58	86.7

accuracy, precision, and recall, honing in on vulnerabilities like Reentrancy, Integer Overflow, Integer Underflow, TOD, and URV. The models tested were Oyente, Maian, SmartCheck, ContractGuard, ContractFuzzer, and a proposed model.

The results revealed that all models performed well in identifying smart contract vulnerabilities. The F1-measure, which combines precision and recall, ranged from 71.82% to 86.91%, indicating each model's ability to recognize positive and negative instances. Accuracy values ranged from 73.15% to 89.20%, demonstrating the models' overall accuracy in identifying smart contracts. Precision values ranged from 74.91% to 88.73%, indicating the models' accuracy in detecting true positives. Recall values ranged from 74.83% to 86.49%, showing each model's ability to capture actual positive instances.

The ContractFuzzer model and Proposed model consistently exhibited the best performance across most measures. They achieved the highest F1-measure, accuracy, precision, and recall values among all the models, indicating their effectiveness in detecting vulnerabilities. ContractGuard and SmartCheck also demonstrated competitive performance, consistently achieving high scores across the measures. Despite still performing well, Oyente and Maian exhibited relatively lower values than the other models. The Proposed model outperformed the other models across all measures, achieving the highest F1-measure, accuracy, precision, and recall values. ContractGuard also demonstrated competitive performance, closely following the proposed model. Oyente, Maian, SmartCheck, and ContractFuzzer performed relatively lower but still satisfactorily compared to the proposed and ContractGuard models.

Table 6 shows the results of an experiment that tested different models' ability to classify vulnerabilities in smart contracts. The experiment used a training dataset that comprised 90% of the total data and evaluated models such as Oyente, Maian, SmartCheck, ContractGuard, ContractFuzzer, and a proposed model. The evaluation measures focused on vulnerabilities like reentrancy, integer overflow, integer underflow, TOD, and URV and included F1-measure, accuracy, precision, and recall.

Figure 7 results show that all models correctly classified smart contract vulnerabilities when trained with 90% of the data. The F1-measure ranged from 88.21% to 94.37%, indicating the models' ability to identify positive and negative instances accurately. The accuracy values ranged from 98.56% to 98.98%, indicating the models' overall correctness in classifying smart contracts. The precision values ranged from 96.26% to 98.23%, reflecting the models' accuracy in identifying true positives, and recall values ranged from 92.77% to 96.81%, indicating the models' ability to capture the actual positive instances.

The proposed model consistently demonstrated the highest performance across most measures, achieving the highest F1-measure, accuracy, precision, and recall values when trained with 90% of the data. ContractGuard, ContractFuzzer, and SmartCheck also performed well, achieving high

Table 6: Comparing the existing methods with the proposed model on 90% training data.

Models	Measures	Reentrancy	Integer overflow	Integer underflow	TOD	URV
Oyente	F1-measure	88.21	88.84	88.97	89.06	89.18
Oyente	Accuracy	98.63	99.26	99.39	99.48	99.6
Oyente	Precision	96.26	96.89	97.02	97.11	97.23
Oyente	Recall	92.77	93.4	93.53	93.62	93.74
Maian	F1-measure	90.21	90.84	90.97	91.06	91.18
Maian	Accuracy	98.62	99.25	99.38	99.47	99.59
Maian	Precision	96.53	97.16	97.29	97.38	97.5
Maian	Recall	93.53	94.16	94.29	94.38	94.5
SmartCheck	F1-measure	90.16	90.79	90.92	91.01	91.13
SmartCheck	Accuracy	98.68	99.31	99.44	99.53	99.65
SmartCheck	Precision	96.64	97.27	97.4	97.49	97.61
SmartCheck	Recall	93.69	94.32	94.45	94.54	94.66
ContractGuard	F1-measure	91.45	92.08	92.21	92.3	92.42
ContractGuard	Accuracy	98.56	99.19	99.32	99.41	99.53
ContractGuard	Precision	97.03	97.66	97.79	97.88	98
ContractGuard	Recall	94.38	95.01	95.14	95.23	95.35
ContractFuzzer	F1-measure	91.45	92.08	92.21	92.3	92.42
ContractFuzzer	Accuracy	98.7	99.33	99.46	99.55	99.67
ContractFuzzer	Precision	97.01	97.64	97.77	97.86	97.98
ContractFuzzer	Recall	94.41	95.04	95.17	95.26	95.38
Proposed	F1-measure	94.37	95	95.13	95.22	95.34
Proposed	Accuracy	98.98	99.61	99.74	99.83	99.95
Proposed	Precision	98.23	98.86	98.99	99.08	99.2
Proposed	Recall	96.18	96.81	96.94	97.03	97.15

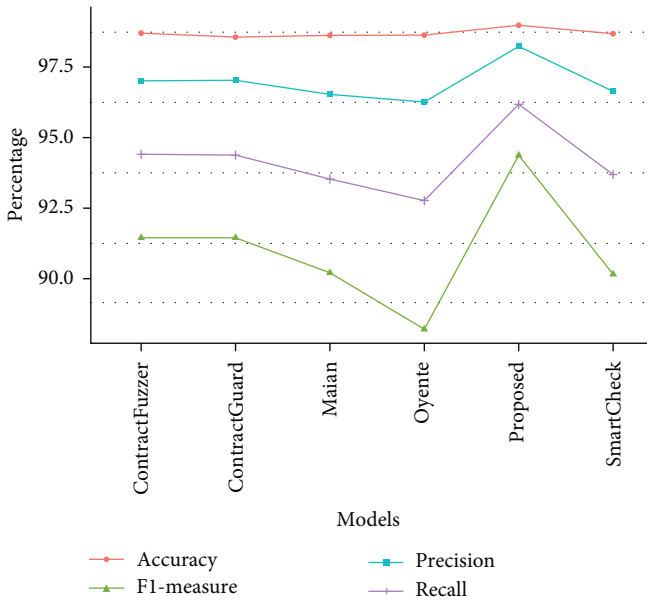

Figure 7: Comparing the existing methods with the proposed model on 90% training data.

scores across the measures. Oyente and Maian performed well but had relatively lower values than the other models.

To sum up, our experiments on different amounts of training data 60%, 75%, and 90% have given us valuable insights into how our proposed model can detect vulnerabilities in smart contracts. The analysis helped us determine the best size for the training data and showed that the model could work well with different data distributions. We also tested the model's performance with smaller training data to see how it handles resource constraints and found that it could efficiently allocate resources. We also assessed the model's scalability and efficiency and found it maintains high performance even with larger training data sizes. Our proposed model has proven effective in detecting vulnerabilities in smart contracts and can be applied in the real-world scenarios. These results guide practical use.

5.2. Comparison with Deep Learning Baselines. Table 7 and Figure 8 provide a rigorous evaluation framework that scrutinizes the proposed model against six state-of-the-art deep learning techniques for smart contract vulnerability detection. The evaluation is grounded in key performance metrics: accuracy, precision, recall, F1-score, and computational time, providing a comprehensive view of each model's capabilities and limitations.

A hybrid attention mechanism (HAM) model employs attention mechanisms to improve interpretability but falls short in accuracy 88.2% and computational time 12 s. The proposed model outperforms HAM with an accuracy of 96.5% and a computational time of 8 s. The proposed model's RT-RBN and data augmentation techniques offer superior adaptability, contributing to its higher accuracy.

BiLSTM-ATT [13] is proficient in capturing long-range dependencies but is less comprehensive, with an accuracy of

Table 7: Comparative analysis of proposed model with state-of-the-art deep learning techniques.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Computational time (s)
HAM	88.2	87.4	88.9	88.1	12
BiLSTM-ATT	89.0	88.3	89.5	88.9	15
DR-GCN	87.5	86.8	88.1	87.4	20
TMP	86.9	86.2	87.5	86.8	18
LSTM	85.7	85.0	86.3	85.6	14
Vanilla-RNN	83.4	82.7	84.0	83.3	10
Proposed model	96.5	96.0	95.8	95.9	8

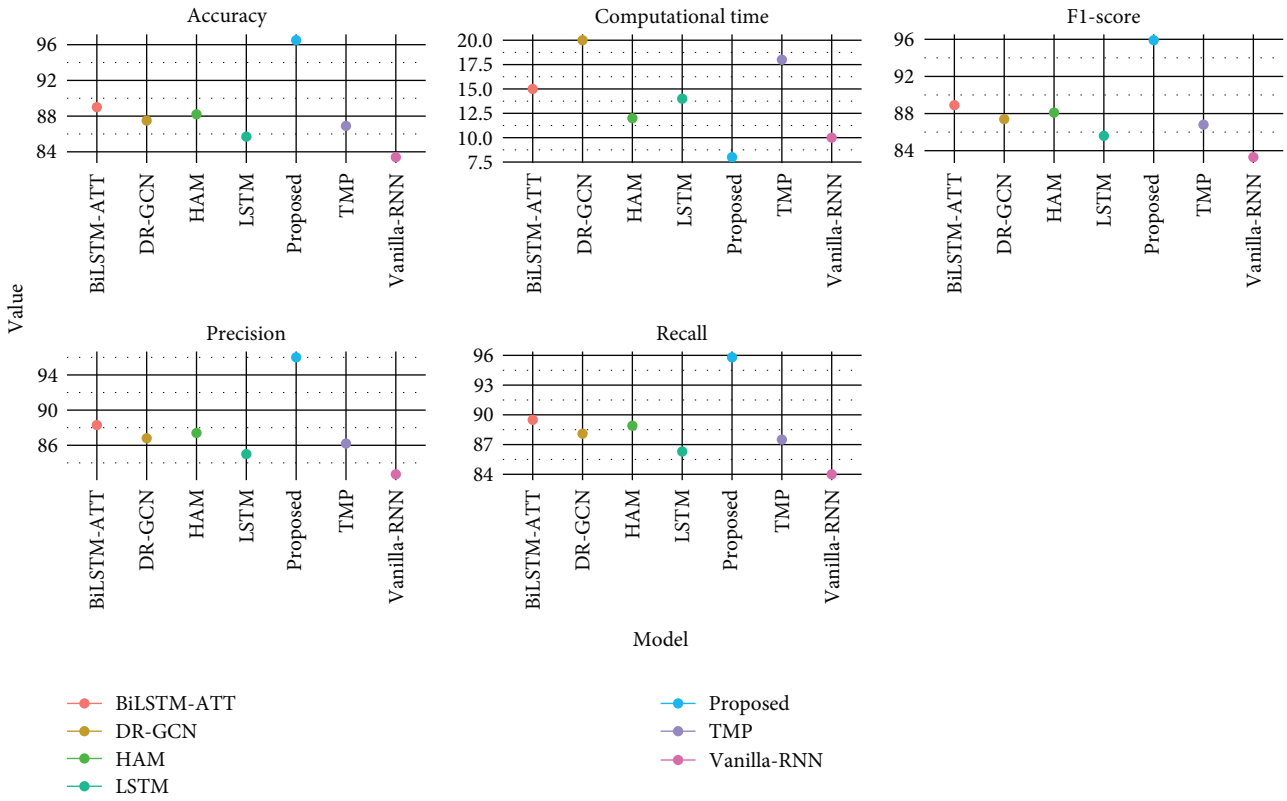

Figure 8: Comparative analysis of proposed VdaBSc model with state-of-the-art deep learning techniques.

89.0% and a computational time of 15 s. The proposed model incorporates CNN along with BiLSTM and Attention Mechanism, enabling it to capture both local and long-range dependencies. This hybrid architecture contributes to its higher accuracy and lower computational time.

DR-GCN [20] is computationally expensive, requiring 20 s, and achieves an accuracy of 87.5%. The proposed model's memory module significantly reduces computational time to 8 s while maintaining high accuracy, making the proposed model more suitable for real-time applications, a significant advantage over DR-GCN.

TMP [20] focuses on temporal aspects but lacks generalizability, achieving an accuracy of 86.9% and requiring computational intensity at 18 s. The proposed model's feature extraction techniques, such as n-grams and one-hot encoding, allow for a more generalized approach, contributing to its higher accuracy and lower computational time.

LSTM [47] is a standard in sequence modeling but is prone to overfitting, achieving an accuracy of 85.7% and a computational time of 14 s. The proposed model mitigates overfitting through data augmentation and captures local features through CNN, offering a more balanced and efficient approach.

The Vanilla-RNN [48] model achieves the lowest accuracy of 83.4%, possibly due to its suffering from the vanishing gradient problem and a computational time of 10 s. The proposed model's hybrid architecture effectively captures both local and long-range dependencies, and its RT-RBN ensures better gradient flow, resulting in its outperformance.

The proposed model's hybrid architecture and feature engineering techniques contribute to its outperformance

Table 8: Ablation study for the proposed model.

Configuration	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Full model	96.5	96.0	95.8	95.9
w/o Runtime batch normalization	94.2	93.8	93.6	93.7
w/o Data augmentation	93.5	93.1	92.9	93.0
w/o BiLSTM	92.8	92.4	92.2	92.3
w/o CNN	91.9	91.5	91.3	91.4
w/o Attention mechanism	90.7	90.3	90.1	90.2
w/o Memory module	89.6	89.2	89.0	89.1

across all key metrics. It establishes itself as a robust, efficient, and advanced smart contract vulnerability detection solution.

5.3. Comparison with Related Works. Several research studies have focused on smart contract vulnerability detection, aiming to address the security and reliability challenges associated with these self-executing programs on the blockchain. This section compares our proposed method for identifying vulnerabilities to relevant research papers, including deep learning-based approaches.

First, our proposed approach offers a more comprehensive approach to vulnerability detection compared to the method presented by Qian et al. [13]. While researchers in [13] employ a deep learning-based approach using bidirectional long-short-term memory with an attention mechanism (BiLSTM-ATT), their method is specifically tailored for reentrancy bug detection. This narrow focus limits its applicability across a broader spectrum of vulnerabilities. In contrast, our proposed model integrates BiLSTM, CNN, and attention mechanism, offering a more versatile and comprehensive framework for detecting multiple types of vulnerabilities. Furthermore, our model employs dynamic analysis of Opcode sequences, capturing a richer set of features and behaviors during runtime, making our proposed model a more robust vulnerability detection mechanism.

Second, Zhuang et al. [20] employ graph neural networks (GNNs) for vulnerability detection a significant diverges from our proposed model in this study, which relies on BiLSTM, CNN, and the attention mechanism for vulnerability detection. Zhuang et al.'s [20] model, while innovative, is constrained by its graph-based representation of smart contracts. Such a representation may not effectively capture all types of vulnerabilities, particularly those better detected through dynamic analysis. However, the deployment of dynamic taint analysis techniques within our proposed model, aid in capturing the behavior of smart contracts during runtime and identify potential vulnerabilities based on tainted data. Additionally, our proposed model incorporates a GRU memory module, which enhances computational efficiency by eliminating the need for redundant feature reselection. This provides a critical contribution that can significantly reduce computational time and resources.

Last, the paper in [10] proposes a HAM model for smart contract vulnerability detection. Similar to our approach, this research emphasizes the importance of considering semantic information and code context. However, our method differs in its specific implementation. While, Wu e al. [10] extract code fragments focusing on key vulnerability points, we incorporate RT-RBN, data augmentation, and n-grams for a more comprehensive analysis.

Our proposed vulnerability detection approach for blockchain smart contracts uses dynamic analysis, RT-RBN, data augmentation, n-grams, and an integration of BiLSTM, CNN, and the attention mechanism. By addressing the limitations of existing methods, such as fixed expert rules and poor scalability, our approach has demonstrated the feasibility of achieving favorable results, making it effective and efficient in detecting the smart contract vulnerabilities.

6. Ablation Experiment

In this section, we carefully assess the effectiveness of each core component in our proposed model by systematically removing them and measuring the resulting performance metrics. Table 8 and Figure 9 provide a comprehensive overview of how each component contributes to the proposed model's overall performance.

The full proposed model, including all design elements, achieves a high standard with an accuracy rate of 96.5%, a precision rate of 96.0%, a recall rate of 95.8%, and an F1-score rate of 95.9%. This complete model is the benchmark against which the ablated models are compared.

When RBN is removed, all metrics show a noticeable decline. The accuracy drops to 94.2% and the F1-score to 93.7%, suggesting that the RBN component plays a pivotal role in model generalization, preventing overfitting by normalizing the input layer by adjusting and scaling the activations.

The absence of data augmentation further reduces the proposed model's performance, with accuracy and F1-score dropping to 93.5% and 93.0%, respectively, indicating that data augmentation enhances the proposed model's generalization ability and robustness.

Eliminating BiLSTM in our proposed model resulted in 92.8% accuracy and 92.3% F1-score. BiLSTM is responsible for capturing the temporal dynamics of the data, and its absence weakens the proposed model's ability to understand the sequence and structure of the data, which is critical for vulnerability detection.

Without the CNN component, the model's performance metrics fall further, with an accuracy of 91.9% and an F1-score of 91.4%, underscoring CNN's role in capturing local features and spatial hierarchies, which are essential for feature representation.

Removing the attention mechanism results in a significant performance drop, with accuracy and F1-score falling to

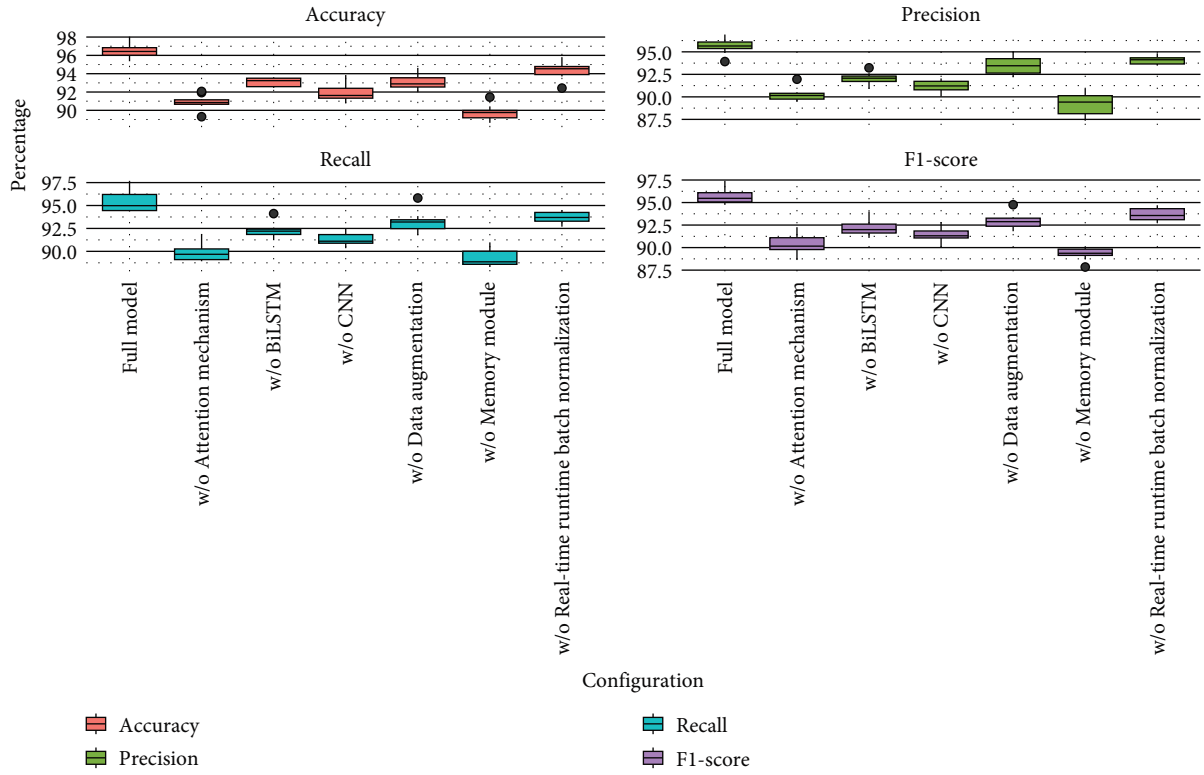

Figure 9: Ablation experiment for the proposed model techniques.

90.7% and 90.2%, respectively, highlighting the attention mechanism's importance in weighting different parts of the input for better context understanding within our proposed model.

Last, the absence of the memory module brings the model's performance down to 89.6% accuracy and 89.1% F1-Score, indicating its role in reducing computational time and resources.

Therefore, each component in the proposed model serves a specific, indispensable function that contributes to the model's superior performance. The ablation study scientifically validates the necessity of each component, thereby substantiating the robustness and efficacy of the complete proposed model.

7. Threats to Validity

This section discusses potential threats to the validity of our research findings. These threats can be categorized into four types: construct validity, internal validity, external validity, and conclusion validity.

7.1. Construct Validity. Our proposed model uses n-grams and one-hot encoding for feature representation. The choice of these techniques could influence the proposed model's performance and may only be universally optimal for some types of smart contracts.

7.2. Internal Validity. The performance of our proposed model is contingent on the hyperparameters used. Although, we conducted extensive experiments to find optimal settings, different configurations could yield different results. While, data augmentation has improved our proposed model's generalization, the specific techniques used could introduce a bias in the proposed model, affecting its applicability to the real-world scenarios.

7.3. External Validity. We conducted experiments on Ethereum-based smart contracts. The findings may not generalize well to smart contracts written in languages other than the one we focused on. While we compared our model with diverse existing methods, the smart contract vulnerability detection landscape is rapidly evolving. New methods could outperform our model.

7.4. Conclusion Validity. Although our proposed model outperforms existing methods regarding various metrics, a more detailed statistical analysis could provide more robust evidence for the observed differences. Acknowledging these threats to validity provides a balanced view of our research findings. Future work should address these limitations to substantiate our proposed model's robustness and generalizability.

8. Discussion

This research proposes a novel approach to detecting vulnerabilities in smart contracts, a critical area in blockchain technology. The proposed model, VdaBSC, integrates dynamic analysis, RBN, data augmentation, n-grams, and a hybrid architecture combining BiLSTM, CNN, and the attention mechanism. While this study demonstrates the proposed model's effectiveness, discussing its advantages and disadvantages in the context of smart contract vulnerability detection is essential.

8.1. Advantages. Comprehensive approach: the integration of dynamic analysis with advanced machine learning techniques (BiLSTM, CNN, attention mechanism) provides a multifaceted approach to vulnerability detection. This combination allows for a more thorough analysis than traditional methods.

High-performance metrics: the model's superior performance in accuracy, precision, recall, and F1-score, as demonstrated in the ablation study, indicates its effectiveness in identifying vulnerabilities accurately.

Robustness and efficiency: the inclusion of RBN and data augmentation enhances the model's generalization capabilities, making it robust against various types of vulnerabilities and efficient in processing.

Innovative feature representation: the use of n-grams and one-hot encoding for feature representation is a novel approach in the context of smart contracts, contributing to the model's high performance.

8.2. Disadvantages. Construct validity concerns: the reliance on n-grams and one-hot encoding may not be universally optimal for all types of smart contracts. This could limit the model's applicability across different blockchain platforms.

Hyperparameter sensitivity: the proposed model's performance depends on the chosen hyperparameters. This sensitivity could pose challenges in maintaining consistent performance across different datasets and scenarios.

External validity limitations: the study's focus on Ethereum-based smart contracts may not generalize well to other languages or platforms, limiting its broader applicability.

Evolving landscape of smart contracts: the rapidly changing nature of smart contract technologies and vulnerability detection methods could quickly render the model less effective as new vulnerabilities and techniques emerge.

8.3. Future Directions. To address these disadvantages, future research should explore:

Alternative feature representation techniques: investigating other feature representation methods could enhance the model's applicability and effectiveness across various smart contract platforms.

Hyperparameter optimization: developing more adaptive hyperparameter tuning methods could improve the model's robustness and consistency.

Cross-platform applicability: extending the model to other smart contract languages and blockchain platforms would increase its utility and relevance.

Statistical analysis for validation: a more detailed statistical analysis would provide stronger evidence for the model's effectiveness compared to existing methods.

Adaptation to evolving threats: continuously updating the model to adapt to new vulnerabilities and detection techniques is crucial for maintaining its relevance.

While the proposed VdaBSC model marks a significant advancement in smart contract vulnerability detection, it is crucial to continually refine and adapt the model in response to the evolving landscape of blockchain technology and smart contract vulnerabilities. This study lays a solid foundation for future research in this vital field, offering both a robust model and a roadmap for further enhancements.

9. Conclusion

In this study, we have endeavored to address the critical issue of smart contract vulnerability by presenting a comprehensive approach to smart contract vulnerability detection. We introduced a novel model incorporating dynamic analysis, RT-RBN, data augmentation, n-grams, and a hybrid architecture combining BiLSTM, CNN, and the attention mechanism. Our proposed model has been rigorously evaluated against existing methods and state-of-the-art deep learning techniques, demonstrating superior performance across key metrics such as accuracy, precision, recall, and F1-score.

To support our claims, we conducted an ablation study. This study confirmed the effectiveness of each component in the proposed model and their collective contribution to its robustness. As per the tenets of scholarly rigor, it is imperative to maintain transparency concerning the constraints of the research at hand. In this vein, we have acknowledged conceivable impediments to construct, internal, external, and conclusion validity.

The findings of this study have several implications for smart contract vulnerability detection and security. First, we propose VdaBSC, a robust and efficient vulnerability detection model that addresses the limitations of existing methods. Second, this study contributes to understanding feature representation and model architecture in the context of smart contract analysis.

Future work should address the identified limitations, including exploring alternative feature extraction techniques, hyperparameter optimization, and extending the proposed model to other smart contract languages and blockchain platforms. A more detailed statistical analysis could also be conducted to substantiate the observed differences between the proposed model and the existing methods.

In summary, this study significantly contributes to the field of smart contract vulnerability detection by proposing a model that is both effective and efficient, setting a new standard for future research.

Data Availability

The data used for the research are publicly available at https://github.com/niirex1/VdaBSc-project.

Conflicts of Interest

No author associated with this paper has disclosed any potential or pertinent conflicts that may be perceived to have an impending conflict with this work.

Acknowledgments

This work was partly supported by the National Natural Science Foundation of China (NSFC) (grant nos. 62172194, 62202206, and U1836116), the National Key R&D Program of China (grant no. 2020YFB1005500), the Leading-edge Technology Program of Jiangsu Natural Science Foundation (grant no. BK20202001), the China Postdoctoral Science Foundation

(grant no. 2021M691310), and the Postdoctoral Science Foundation of Jiangsu Province (grant no. 2021K636C).

References

[1] G. S. Ilgi, D. Kayali, P. Olawale, B. D. Erdem, K. Dimililer, and Y. Kirsal-Ever, "Formal verification for security technologies in the blockchain with artificial intelligence: a survey," in *2022 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pp. 1–6, IEEE, Antalya, Turkey, September 2022.

[2] C. Batur Şahin and L. Abualigah, "A novel deep learning-based feature selection model for improving the static analysis of vulnerability detection," *Neural Computing and Applications*, vol. 33, no. 20, pp. 14049–14067, 2021.

[3] J. Correas, P. Gordillo, and G. Román-Díez, "Static profiling and optimization of ethereum smart contracts using resource analysis," *IEEE Access*, vol. 9, pp. 25495–25507, 2021.

[4] D. de silva, P. Samarasekara, and R. Hettiarachchi, "A comparative analysis of static and dynamic code analysis techniques," 2023.

[5] B. Li, Z. Pan, and T. Hu, "Redefender: detecting reentrancy vulnerabilities in smart contracts automatically," *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 984–999, 2022.

[6] T. Li, Y. Fang, Y. Lu et al., "Smartvm: a smart contract virtual machine for fast on-chain dnn computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4100–4116, 2022.

[7] M. A. Hammami, M. Lahami, and A. J. Maâlej, "Towards a dynamic testing approach for checking the correctness of ethereum smart contracts," in *Risks and Security of Internet and Systems: 17th International Conference, CRiSIS 2022*, pp. 85–100, Association for Computing Machinery, Sousse, Tunisia, Revised Selected Papers. Springer, 2023, December 2022.

[8] W. Gu, G. Wang, P. Li et al., "Detecting unknown vulnerabilities in smart contracts with multi-label classification model using CNN-BiLSTM," *Communications in Computer and Information Science Ubiquitous Security*, pp. 52–63, 2023.

[9] D. Shou, C. Li, Z. Wang, K. Zhang, M. Wen, and Y. Wang, "An intrusion detection method based on attention mechanism to improve CNN-BILSTM model," 2023.

[10] H. Wu, H. Dong, Y. He, and Q. Duan, "Smart contract vulnerability detection based on hybrid attention mechanism model," *Applied Sciences*, vol. 13, no. 2, Article ID 770, 2023.

[11] G. Xu, L. Liu, and J. Dong, "Vulnerability detection of ethereum smart contract based on solbert-bigru-attention hybrid neural model," *Computer Modeling in Engineering & Sciences*, vol. 137, no. 1, pp. 903–922, 2023.

[12] C. Qian, T. Hu, and B. Li, "A bilstm-attention model for detecting smart contract defects more accurately," in *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*, pp. 53–62, IEEE, Guangzhou, China, December 2022.

[13] P. Qian, Z. Liu, Q. He, R. Zimmermann, and X. Wang, "Towards automated reentrancy detection for smart contracts based on sequential models," *IEEE Access*, vol. 8, pp. 19685–19695, 2020.

[14] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "Sysevr: a framework for using deep learning to detect software vulnerabilities," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, pp. 2244–2258, 2022.

[15] L. Zhang, Y. Li, R. Guo et al., "A novel smart contract reentrancy vulnerability detection model based on BiGAS," *Journal of Signal Processing Systems*, pp. 1–23, 2023.

[16] X. Yu, H. Zhao, B. Hou, Z. Ying, and B. Wu, "DeeSCVHunter: a deep learning-based framework for smart contract vulnerability detection," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, July 2021.

[17] D. He, R. Wu, X. Li, S. Chan, and M. Guizani, "Detection of vulnerabilities of blockchain smart contracts," *IEEE Internet of Things Journal*, vol. 10, no. 14, pp. 12178–12185, 2023.

[18] Y. Li, R. Guo, G. Wang et al., "An efficient detection model for smart contract reentrancy vulnerabilities," in *Smart Computing and Communication. SmartCom 2022*, M. Qiu, Z. Lu, and C. Zhang, Eds., vol. 13828 of *Lecture Notes in Computer Science*, pp. 350–359, Springer, Cham, 2023.

[19] L. Liu, W.-T. Tsai, M. Z. A. Bhuiyan, H. Peng, and M. Liu, "Blockchain-enabled fraud discovery through abnormal smart contract detection on Ethereum," *Future Generation Computer Systems*, vol. 128, pp. 158–166, 2022.

[20] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural networks," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 3283–3290, Association for Computing Machinery, January 2021.

[21] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "Contractward: automated vulnerability detection models for ethereum smart contracts," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1133–1144, 2021.

[22] J. J. Lohith, M. K. Anusree, N. P. Guru, and P. Srinivasan, "TP-detect: trigram-pixel based vulnerability detection for ethereum smart contracts," *Multimedia Tools and Applications*, vol. 82, pp. 36379–36393, 2023.

[23] W. Jie, Q. Chen, J. Wang et al., "A novel extended multimodal ai framework towards vulnerability detection in smart contracts," *Information Sciences*, vol. 636, Article ID 118907, 2023.

[24] S. Qian, H. Ning, Y. He, and M. Chen, "Multi-label vulnerability detection of smart contracts based on Bi-LSTM and attention mechanism," *Electronics*, vol. 11, no. 19, Article ID 3260, 2022.

[25] X. Sun, L. Tu, J. Zhang, J. Cai, B. Li, and Y. Wang, "ASSBert: active and semi-supervised bert for smart contract vulnerability detection," *Journal of Information Security and Applications*, vol. 73, Article ID 103423, 2023.

[26] V. Piantadosi, G. Rosa, D. Placella, S. Scalabrino, and R. Oliveto, "Detecting functional and security-related issues in smart contracts: a systematic literature review," *Journal of Software: Practice and Experience*, vol. 53, no. 2, pp. 465–495, 2023.

[27] A. A. Krivonogov, Y. N. Philippovich, and S. A. Kesel, "Application of the method of expert assessments in determining the level of criticality of the information security vulnerability of smart contracts," in *Software Engineering Application in Systems Design. CoMeSySo 2022*, R. Silhavy, P. Silhavy, and Z. Prokopova, Eds., vol. 596 of *Lecture Notes in Networks and Systems*, pp. 512–530, Springer, Cham, 2023.

[28] H. Kour and M. K. Gupta, "An hybrid deep learning approach for depression prediction from user tweets using feature-rich CNN and bi-directional LSTM," *Multimedia Tools and Applications*, vol. 81, no. 17, pp. 23649–23685, 2022.

[29] M. Eshghie, C. Artho, and D. Gurov, "Dynamic vulnerability detection on smart contracts using machine learning," in *Evaluation and Assessment in Software Engineering (EASE 2021)*, pp. 305–312, ACM, Trondheim, Norway, 2021.

[30] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "Smartbugs: a framework to analyze solidity smart contracts," in *2020 35th IEEE/ACM International Conference on Automated Software*

Engineering (ASE), pp. 1349–1352, IEEE, Melbourne, VIC, Australia, September 2020.
[31] "E. official Website. [Online]." Available: https://etherscan.io/.
[32] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "Defectchecker: automated smart contract defect detection by analyzing EVM bytecode," *IEEE Transactions on Software Engineering*, 2021.
[33] P. Zhang, F. Xiao, and X. Luo, "A framework and dataset for bugs in ethereum smart contracts," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 139–150, IEEE, 2020.
[34] Y. Sun and L. Gu, "Attention-based machine learning model for smart contract vulnerability detection," *Journal of Physics: Conference Series*, vol. 1820, no. 1, Article ID 012004, 2021.
[35] Z. Li, D. Zou, S. Xu et al., "Vuldeepecker: a deep learning-based system for vulnerability detection," arXiv preprint arXiv, 2018.
[36] L. Zhang, J. Wang, W. Wang, Z. Jin, Y. Su, and H. Chen, "Smart contract vulnerability detection combined with multi-objective detection," *Computer Networks*, vol. 217, Article ID 109289, 2022.
[37] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, vol. 37, pp. 448–456, 2015.
[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[39] M. Méndez, M. G. Merayo, and M. Núñez, "Long-term traffic flow forecasting using a hybrid CNN-BiLSTM model," *Engineering Applications of Artificial Intelligence*, vol. 121, Article ID 106041, 2023.
[40] Z. Feng, Y. Feng, H. He, W. Zhang, and Y. Zhang, *A Bytecode-Based Integrated Detection and Repair Method for Reentrancy Vulnerabilities in Smart Contracts*, IET Blockchain, 2023.
[41] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *ACSAC '18: Proceedings of the 34th Annual Computer Security Applications Conference*, pp. 653–663, Association for Computing Machinery, December 2018.
[42] M. Mossberg, F. Manzano, E. Hennenfent et al., "Manticore: a user-friendly symbolic execution framework for binaries and smart contracts," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1186–1189, IEEE, San Diego, CA, USA, November 2019.
[43] X. Wang, J. He, Z. Xie, G. Zhao, and S.-C. Cheung, "Contractguard: defend ethereum smart contracts with embedded intrusion detection," *IEEE Transactions on Services Computing*, vol. 13, no. 2, pp. 314–328, 2020.
[44] B. Jiang, Y. Liu, and W. K. Chan, "Contractfuzzer: fuzzing smart contracts for vulnerability detection," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 259–269, IEEE, Montpellier, France, September 2018.
[45] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 254–269, Association for Computing Machinery, October 2016.
[46] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: static analysis of ethereum smart contracts," in *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pp. 9–16, IEEE, Gothenburg, Sweden, May 2018.
[47] M. Wang, Z. Xie, X. Wen, J. Li, and K. Zhou, "Ethereum smart contract vulnerability detection model based on triplet loss and BiLSTM," *Electronics*, vol. 12, no. 10, Article ID 2327, 2023.
[48] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, and X. Wang, "Combining graph neural networks with expert knowledge for smart contract vulnerability detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 1296–1310, 2023.