JOURNAL OF INFORMATION SCIENCE AND ENGINEERING XX, XXXX-XXXX (2024)
DOI: XX.XXX.XXXX.XX

A Vulnerability Detection Scheme for Smart Contracts Based on Large Language Models and Deep Learning Techniques*

YUNHAO ZHAO[1], HAO ZHANG[1], LIANJIE WANG[1],
Keqing Wang[1], Wanshui Song[2], Zhongwen Zhang[1]
Wenyin Zhang[1,+] and Yeh-Cheng Chen[3]
[1] School of Information Science and Engineering, Linyi University, Linyi, 276000, China
[2] College of Computer Science and Technology, Harbin Engineering University, Harbin, 150000, China
[3] Department of Computer Science, University of California Davis, Davis, 94555, USA
E-mail: zhangwenyin@lyu.edu.cn

The identification of vulnerabilities in smart contracts is a vital component of blockchain security technology. Traditional techniques such as static analysis, fuzz testing, and formal verification often have elevated false-positive rates and lack adaptive learning capacities. This study presents GPT-CodeBert Deep Learning Vulnerabilities Identification Architecture, a tool for detecting vulnerabilities in smart contracts using deep learning. The system begins by utilizing big language models, specifically ChatGPT, to examine smart contracts and identify essential functions that may conceal vulnerabilities. The CodeBert model is consequently employed for feature extraction on these functions. A high-precision predictive model is eventually developed by integrating an attention mechanism with a bidirectional long short-term memory network. The experimental results imply that the suggested framework is efficient, extremely accurate, and possesses superior detection rates regarding accuracy, precision, and recall. The proposed framework achieves an accuracy of 96% in identifying reentrancy vulnerabilities and 98% in recognizing Tx-Origin issues. Keywords: ChatGPT, CodeBert, Smart Contracts, Deep Learning

Keywords: ChatGPT, CodeBert, BiLSTM, Smart Contracts, Deep Learning

1. INTRODUCTION

The fast advancement and wide utilization of blockchain technology [1] have made smart contracts a crucial element that can broaden blockchain's potential while introducing unparalleled hurdles. The intrinsic security flaws of smart contracts have led to considerable financial losses and eroded trust in the Bitcoin ecosystem. However, major security holes have been observed [2] previously, such as the reentrancy attack on the DAO [3] and the default visibility vulnerability in the Parity MultiSig Wallet. These holes have cost a large amount of money and hurt the credibility of blockchain technology. These occurrences highlight the pressing need for further research into the security of smart contracts.

Received March 11, 2014; revised June 20, 2014; accepted September 28, 2014.
+Corresponding author: zhangwenyin@lyu.edu.cn
Communicated by the editor.

In the field of smart contract vulnerability detection, present technologies are largely categorized into three [4]: traditional detection methods [5], deep learning-based methods [6], and methods utilizing large language models such as ChatGPT. Traditional techniques, such as fuzz testing [7], formal verification [8], and symbolic execution [9], have exhibited practical efficacy. Nevertheless, their dependence on specialized knowledge and minimal automation restrains their adaptability to the constantly changing environment of risks. Deep learning methodologies [10–17], through autonomous feature learning of smart contract vulnerabilities, have reduced reliance on expert knowledge and enabled novel approaches for automated vulnerability detection. However, managing massive sample quantities and intricate smart contract codes has considerably hindered the successful extraction of vulnerability-related features. Despite ChatGPT's robust proficiency in code comprehension, its direct utilization for smart contract vulnerability detection may not provide optimal outcomes [18, 19].

This study presents a code trimming strategy utilizing ChatGPT to identify essential code portions that may include vulnerability issues. This method evidently enhances sample quality and detection efficacy. The refined CodeBert model [20, 21] is combined with an attention-augmented Bidirectional Long Short-Term Memory (BiLSTM) deep learning model, and this approach results in the effective, accurate identification of smart contract vulnerabilities. This study's principal contributions are listed below:

- ChatGPT is employed for pruning operations to isolate the essential functionalities and consequently enhance the semantic density of vulnerability traits. This approach efficiently streamlines smart contract programming by detecting and segregating functions that may possess substantial vulnerabilities. The procedure greatly decreases interference from extraneous code and hence improves the efficiency and precision of ensuing deep learning models.

- The GPT-CodeBert Deep Learning Vulnerabilities Identification Architecture (GCB-DVIA) framework, an advanced approach for detecting vulnerabilities in smart contracts that integrates big language models with deep learning methodologies, is employed. The system efficiently and accurately detects smart contract vulnerabilities by integrating ChatGPT's semantic comprehension, CodeBert's code understanding, and BiLSTM's sequence feature extraction capabilities.

- Comprehensive experiments across several datasets reinforce the efficacy and superiority of the proposed system in detecting vulnerabilities in smart contracts. The detection rates for reentrancy and transaction-origin vulnerabilities are 96% and 98%, respectively, which greatly surpass those of current detection methods.

2. Related Work

Research on vulnerability detection methods is crucial in the domain of smart contract security. Current research can be classified into three categories according to the technological methodologies utilized: conventional detection techniques, deep learning-based detection techniques, and detection techniques leveraging big language models (e.g., ChatGPT).

2.1 Traditional detection method

Conventional approaches primarily depend on a range of static and dynamic analysis techniques. Grieco et al. [7] present Echidna, a fuzzing tool tailored for Ethereum smart contracts. The tool enhances the generation of test cases specific to the distinct functionalities of smart contracts and can automatically identify and exploit particular contract behavior patterns. Park et al. [8] created VeriSmart for the detection of security vulnerabilities in smart contracts through the application of formal verification methods. VeriSmart enhances the efficiency and accuracy of the formal verification process through the adoption of advanced abstractive and optimized solvers, facilitating secure verification of complex smart contracts. Mueller et al. [9] introduced Mythril, an open-source tool for analyzing the security of Ethereum smart contracts, employing symbolic execution for the examination of smart contract code. Mythril effectively identifies deeper vulnerabilities and complex attack vectors by employing an efficient symbolic execution strategy and conducting thorough analyses of contracts. Feist et al. [22] introduce Slither, a static analysis framework designed to convert smart contract code into an intermediate representation. Slither enhances the speed and accuracy of vulnerability detection through optimizations in static analysis and intermediate representations, simultaneously minimizing false positives. The dependence on expert knowledge and the minimal automation of traditional methods restrict their capacity to adjust to the evolving vulnerability landscape.

2.2 Vulnerability detection method based on deep learning

Artificial intelligence has facilitated the detection of vulnerabilities in smart contracts through deep learning techniques. The remarkable feature extraction capability of deep neural networks is employed to autonomously identify patterns of code vulnerabilities. Huang et al. [10] devised a mechanism for detecting vulnerabilities in smart contracts based on multi-task learning. The model enhanced contract vulnerability correlation by parallelizing detection operations within a framework. Liu et al. [11] devised a technique for detecting vulnerabilities in smart contracts through the application of deep learning and expert rules. The enhancement of vulnerability identification and smart contract security verification is achieved through the improved accuracy and efficiency of deep learning's pattern recognition and expert rules. Sendner et al. [12] propose employing deep transfer learning to detect vulnerabilities in smart contracts [23]. This approach employs pre-existing data sets for model training using transfer learning, improving detection in data-scarce situations and strengthening model generalization and efficiency. Sun et al. [13] developed ASSBert, a mechanism for detecting smart contract vulnerabilities utilizing active and semi-supervised learning. This approach improves vulnerability detection with minimal annotation data and provides a technology framework for analyzing smart contract security. Notwithstanding these advancements, deep learning methods encounter difficulties in feature extraction and the encoding of complex semantic information.

2.3 Large Language Model Based Vulnerability Detection Method

Researchers are examining vulnerability detection systems based on huge language models to resolve the aforementioned challenges. ChatGPT's proficiency in code comprehension positions it as a possible detector of smart contract vulnerabilities. Sun et al. [18]

introduced GPTScan, an innovative method for identifying logical errors in smart contracts, by integrating GPT technology with program analysis. This approach enhances vulnerability detection by integrating ChatGPT with program analysis, hence furthering research in smart contract security. Hu et al. [19] present the GPTLens framework for identifying vulnerabilities in smart contracts through the utilization of large language models. The system employs a two-stage generative and discriminative approach to enhance the accuracy and efficiency of ChatGPT vulnerability identification. Nonetheless, these methods possess limitations. Expertise is required to formulate vulnerability situations and attributes in GPTScan, constraining its ability to respond to swift vulnerability advancements. Any misconfiguration of scenarios and attributes might significantly diminish the detection rate. GPTLens utilizes ChatGPT, and its training data includes substantial non-smart contract vulnerability information, which impacts its detection efficiency.

To tackle the aforementioned difficulties, we suggest the GCB-DVIA methodology. In contrast to conventional approaches that depend on expert experience, GCB-DVIA employs the semantic information of vulnerabilities for training and prediction via a deep learning model. Initially, ChatGPT is employed to analyze the code, concentrating on the critical aspects of vulnerability risk, thereby enhancing the focus on the semantic information related to internal vulnerabilities, minimizing distractions, and augmenting the specificity and precision of vulnerability detection. We subsequently refine the CodeBert model to tailor it for the extraction of vulnerability aspects from smart contracts written in Solidity. GCB-DVIA achieves efficient and precise identification of smart contract vulnerabilities by integrating the fine-tuned CodeBert model with the BiLSTM deep learning model featuring an attention mechanism. This solution utilizes transfer learning to swiftly adapt to new vulnerabilities and successfully address the rapid developments in smart contract security. This adaptive and efficient approach offers a more sophisticated solution for the security detection of smart contracts.

3. GCB-DVIA FRAMEWORK

We present the GCB-DVIA framework, a novel deep learning architecture explicitly developed for the identification of smart contract vulnerabilities, with its primary structure and method illustrated in Figs.1 and 2. The framework is the inaugural solution to utilize ChatGPT for the extraction of smart contract features and comprises three essential stages:

- **Data Preprocessing**: At this stage, we utilize a refined iteration of ChatGPT to pinpoint and extract essential code snippets pertinent to security vulnerabilities in smart contracts. This technique maximally utilizes ChatGPT's robust semantic comprehension skills [24], significantly diminishing the impact of extraneous information and facilitating the accurate identification of potential vulnerability feature functions.

- **Feature Extraction**: We employ CodeBert [25] to do a comprehensive analysis of the code segments obtained in the initial phase, deriving high-dimensional feature values that signify vulnerability attributes. CodeBert excels in its deep comprehension of programming languages, allowing it to accurately discern patterns and

characteristics associated with vulnerabilities in intricate code architectures.

- **Vulnerability Detection and Classification**: In the concluding phase, a BiLSTM model combined with a self-attention mechanism is employed for vulnerability categorization and prediction.

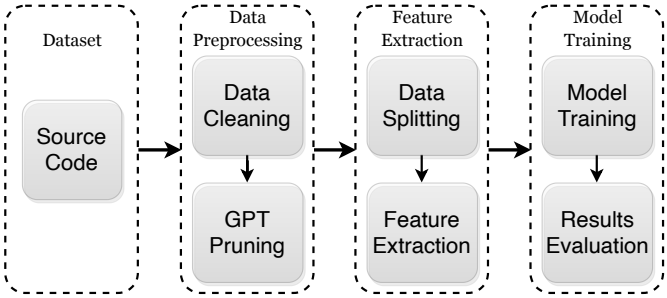

Fig. 1. This figure illustrates the architecture of the GCB-DVIA framework, comprising three main stages: data preprocessing, feature extraction, and model training. The framework integrates ChatGPT for code pruning, CodeBert for feature extraction, and BiLSTM for vulnerability classification, enabling efficient and accurate detection.

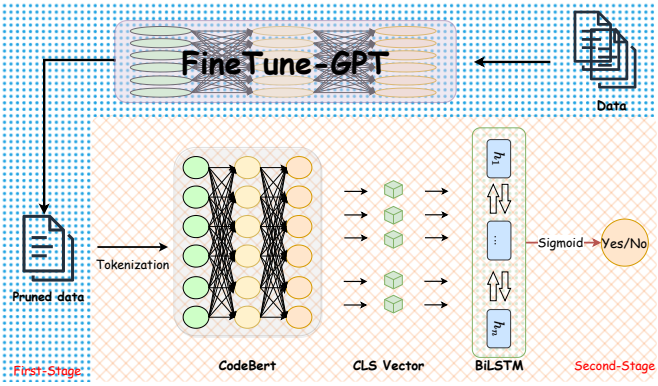

Fig. 2. This figure depicts the workflow of the GCB-DVIA framework, covering data preprocessing, feature extraction with Fine-CodeBert, and vulnerability classification using BiLSTM with attention mechanisms.

This framework's primary novelty is the amalgamation of ChatGPT with deep learning methodologies, facilitating rapid and precise vulnerability detection in smart contract security analysis. This multidisciplinary technology integration enhances the effectiveness of vulnerability identification and broadens the research methodology and application scope in smart contract security analysis.

3.1 Data preprocessing

The data preprocessing stage emphasizes the extraction and cleansing of source code to enhance data quality and assure its appropriateness for further analysis. Fig.3 illustrates that the entire process is segmented into three primary stages: data cleansing, contract segmentation, and GPT filtration.

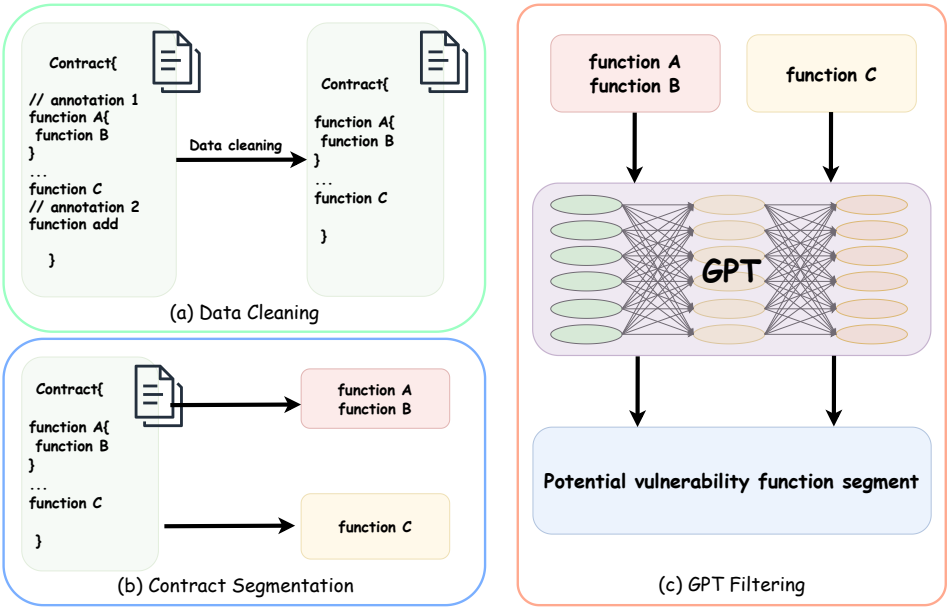

Fig. 3. This figure details the data preprocessing workflow, consisting of three main steps: (a) cleaning redundant data, (b) segmenting long code snippets, and (c) pruning irrelevant segments using ChatGPT. These steps ensure high-quality and relevant input for further analysis.

During the data cleaning procedure, as seen in step (a) of Fig.3, we first employ regular expressions to exclude comments and other unstructured characters from the source code. This not only formats the code but also minimizes its size, enhancing the efficiency of further analysis and filtering with ChatGPT. Subsequently, due to the frequent structural similarities across common code libraries in the source code (as depicted in Fig.4), we established a blacklist of these libraries. The function names in the contract are juxtaposed with those in the blacklist, and matching functions are eliminated to minimize interference in later phases.

During contract segmentation, as illustrated in procedure b of Fig.3. We account for ChatGPT's input token limitation (maximum of 16,385 tokens) by dividing the excessively lengthy contract code by function while maintaining the interrelation between functions. For instance, if function A invokes function B, we ensure that both functions reside within the same code section, as depicted in Fig.3. Furthermore, due to CodeBert's input restriction of 512 tokens, code snippets shorter than 512 tokens are omitted in the subsequent ChatGPT filtering to maintain process efficacy.

In the GPT filtering procedure, as seen in step (c) of Fig.3, we initially refine Chat-

```
library SafeMath {
    function add(uint256 a, uint256 b)internal pure returns(uint256) {
        uint256 c=a+ b;
        require(c>=a,"safeMath: addition overflow");
        return c;
    }
}
```

Fig. 4. This figure presents an example of removing redundant library code during data preprocessing, ensuring the dataset focuses on relevant contract logic for vulnerability detection.

GPT. The fine-tuning procedure is depicted in Fig.5. We initially chose 128 smart contracts manually as valid scenarios. Various probable correct vulnerability code portions were detected in each contract. Reentrancy vulnerabilities typically manifest when a contract permits re-entry into the same function prior to the completion of an external call, resulting in several invocations. In the filtering process, we concentrated on the rationality of external function calls and their corresponding state modifications, specifically assessing whether transfer operations were finalized prior to an external contract invocation. We meticulously identified the appropriate probable vulnerability code parts from each contract and utilized these accurate instances to refine ChatGPT. Consequently, by persistently refining the output format and precision of ChatGPT, the results more closely conformed to our unique requirements.

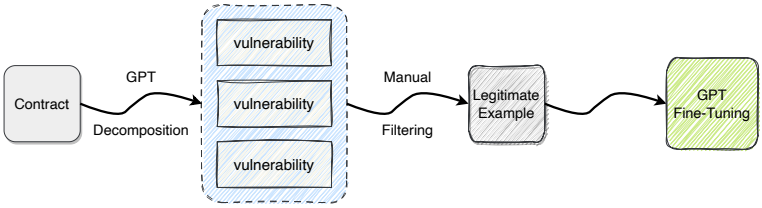

Fig. 5. This figure illustrates the fine-tuning process of ChatGPT, optimizing its ability to identify vulnerability-related code segments while minimizing irrelevant data interference.

The primary objective of fine-tuning is to enhance ChatGPT's ability to effectively analyze smart contract code, particularly with vulnerability filtering, while minimizing the removal of essential data. The fine-tuning method includes modifying the hyperparameters of ChatGPT, as illustrated in Table 1, to yield more accurate results when identifying potentially vulnerable functions. The outputs are presented in JSON format, as illustrated in Fig.6, and encompass essential information such the names and types of state variables, function names, and their visibility. This organized output allows us to easily identify and extract the essential code segments. The information acquired via ChatGPT is utilized to precisely extract pertinent code snippets from the sanitized source code repository. The cleaned smart contract function snippets were acquired using these procedures, resulting

in a dataset size reduction to 31% of the original dataset.

Table 1. ChatGPT Hyperparameter Configuration for Vulnerability Filtering Optimization.

Hyperparameter Name	value	Purpose
Batch size	8	Affects the number of samples per parameter update.
Learning rate multiplier	0.1	Controls the step size for parameter updates.
Number of epochs	5	Determines the number of times the model is trained.

```
{
"state variables": [
    {
      "name": "tokenDestroyed",
      "type": "uint256",
      "visibility":"public"
    }
  ],
"functions_to_audit":[
    {
      "name":"_transfer",
      "potential_vulnerabilities":[
          "unchecked-transfe"
      ]
    }
  ]
}
```

Fig. 6. This figure shows ChatGPT's output in JSON format, extracting critical state variables and functions.

3.2 Feature extraction

We adopted the CodeBert model for feature extraction to facilitate the training and prediction tasks of smart contract vulnerability detection [26]. As CodeBert was not originally trained on the Solidity language, we fine-tuned the model for this specific application, and the modified model is designated as Fine-CodeBert.

Initially, we tokenize the smart contracts employing the CodeBert tokenizer. Due to the model's restricted input capacity, it is impractical to directly input lengthy sentences, such as a full source code file. Consequently, we employ a sliding window technique with a 256-token window to segment the code. The quantity of created input segments is thereafter compared to the established threshold (14 segments) to ascertain whether to incorporate padding segments or to terminate surplus segments. Incomplete segments are supplemented with designated padding tokens, while excess segments are trimmed, thereby maintaining uniformity in the dimensions of the input data. The tokenized data from each smart contract $\mathbf{X} \in \mathbb{R}^{1\times14\times512}$ serves as input for the Fine-CodeBert model.

Subsequently, we do feature extraction. In the Fine-CodeBert model, every input sequence commences with a [CLS] token. Upon processing the input through many levels,

the ultimate state of the [CLS] token serves as the condensed representation of the entire input sequence, effectively encapsulating the fundamental semantics of the input data. Consequently, we obtain the [CLS] token from each segmented sequence as the feature representation of the smart contract $\mathbf{X} \in \mathbb{R}^{1\times 768}$. The extraction procedure is depicted in Fig.7.

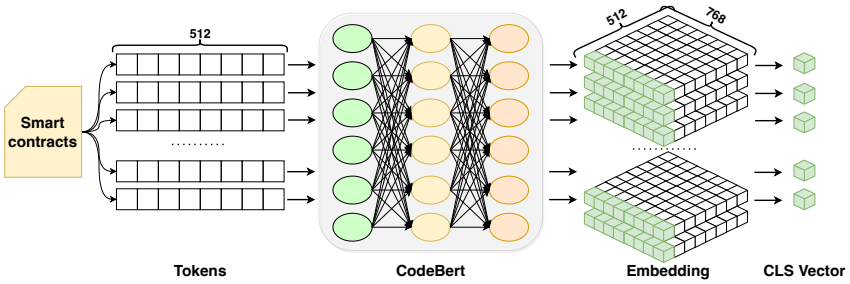

Fig. 7. This figure demonstrates the feature extraction process using CodeBert. Tokenized inputs are processed, and the [CLS] token represents the semantic features of the code for downstream tasks.

3.3 Model Training

Subsequent to feature extraction, we utilize BiLSTM network augmented with a self-attention mechanism for model training and prediction [27]. The model's input is $\mathbf{X} \in \mathbb{R}^{B\times 768}$, with the primary design illustrated in Fig.8. The BiLSTM proficiently gathers contextual information from code sequences in both forward and backward directions. The incorporation of this contextual information improves forecast accuracy. The self-attention method allows the model to concentrate on the more important portions of code while ignoring those of lesser semantic significance, therefore enhancing the model's overall performance.

The BiLSTM design consists of two separate LSTM layers: one processes the sequence in a forward direction, and the other processes it in reverse. Each LSTM layer comprises numerous LSTM units, each equipped with three gates and a cell state, which collectively provide the retention of information continuity and the proficient learning from extended sequence data:

$$\begin{aligned}
\text{Input Gate:} \quad & i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\text{Forget Gate:} \quad & f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
\text{Output Gate:} \quad & o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
\text{Candidate Cell State:} \quad & \tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
\text{Cell State Update:} \quad & c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \\
\text{Hidden State Update:} \quad & h_t = o_t \cdot \tanh(c_t)
\end{aligned} \tag{1}$$

In these equations, σ denotes the sigmoid activation function, which compresses any real number into the interval (0, 1). tanh, the hyperbolic tangent activation function, outputs values in the range (-1, 1).W_f, W_i, W_c, W_o are the weight matrices corresponding

to the forget gate, input gate, candidate cell state, and output gate, respectively. b_f, b_i, b_c, b_o are the bias terms for the forget gate, input gate, candidate cell state, and output gate, respectively. h_{t-1} denotes the hidden state from the previous timestep, x_t represents the input vector at the current timestep, and c_{t-1} is the cell state from the previous timestep. c_t and h_t are the updated cell state and hidden state at the current timestep, respectively.

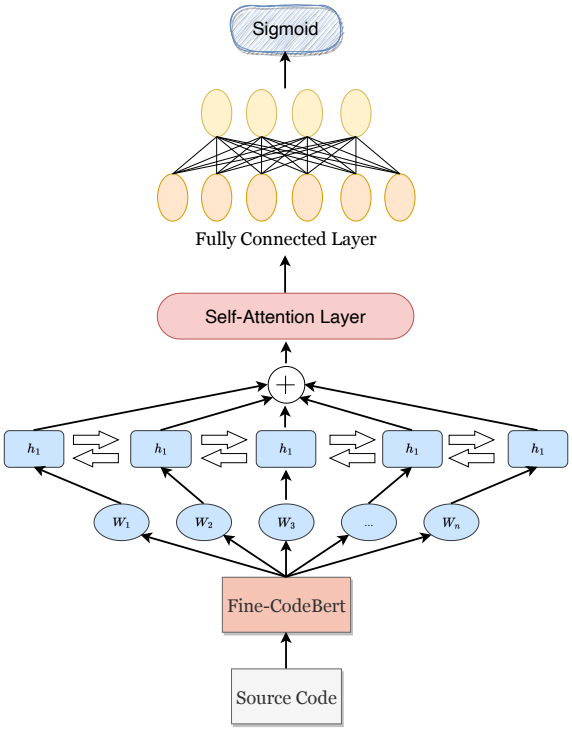

Fig. 8. This figure depicts the BiLSTM model combined with an attention mechanism, capturing contextual information to improve the detection of complex vulnerabilities.

4. Experiments and Analysis

4.1 Experimental Setup

4.1.1 Dataset

In this study, smart contracts implemented on the Ethereum platform are investigated [28]. An in-depth analysis of four prevalent, representative vulnerabilities identified by the Smart Contract Vulnerability Classification Registry was conducted [29]: reentrancy (RENT) vulnerability, tx.origin authentication (TX) vulnerability, unchecked return (UR) value, and locked fund (LE) vulnerability.

Two publicly accessible Ethereum smart contract datasets were used: the ScrawlD dataset [30] and the Slither Audited Smart Contracts dataset [31]. Both datasets comprise

authentic smart contracts deployed on the Ethereum blockchain. The ScrawlD dataset has 9,123 samples, whereas the Slither Audited Smart Contracts dataset encompasses 120,608 samples.These datasets were selected for their complementary characteristics: the ScrawlD dataset, with its smaller and more focused sample size, is ideal for validating the detection capabilities on specific vulnerabilities, while the larger Slither dataset provides a robust foundation for model training and enhances generalization across a diverse range of vulnerabilities.

To address the problem of sample imbalance, samples were selectively extracted from the Slither Audited Smart Contracts dataset according to several categories of vulnerabilities to create a balanced dataset. The minimum ratio of positive to negative samples was at least 1:3. The quantity of samples in each of the four assembled datasets is specified in Table 2.

The detection outcomes of the proposed strategy relative to existing technologies utilizing the ScrawlD dataset were assessed. This dataset was evaluated utilizing tools including Slither, Mythril, Smartcheck, Oyente, and Osiris, with a specific focus on assessing the identification of vulnerabilities such as RENT, Ether locking, and TX concerns. The amounts of these three categories of vulnerabilities are presented in Table 3.

Table 2. Number of Samples in Datasets Constructed According to Different Types of Vulnerabilities.

Vulnerability Dataset	Positive Samples	Negative Samples	Total
TX Dataset	3,043	6,086	9,129
LE Dataset	3,334	6,666	10,000
RENT Dataset	10,204	21,030	31,234
UR Dataset	42,573	78,035	120,608

Table 3. Quantities of Different Types of Vulnerabilities in the ScrawlD Dataset.

Vulnerability Type	Positive Samples	Negative Samples
TX	283	8,840
LE	1,696	7,427
RENT	5,132	3,991

4.1.2 Experimental Environment

This study utilized an environment comprising Ubuntu 22.04 as the operating system, Python 3.10 as the programming language, CUDA 12.1, and PyTorch 2.1.0.All experiments were performed on a high-performance machine with an Intel(R) Xeon(R) Gold 6430 processor comprising 16 vCPUs, a single NVIDIA RTX 4090 (with 24GB of VRAM), and 120GB of RAM.

4.1.3 Parameter Settings

The parameter configuration for model training and testing was defined to improve performance and guarantee result reliability. The batch size was configured to 2,048 to optimize the memory requirements of training alongside processing efficiency. Binary

cross-entropy loss was utilized because it is appropriate for binary classification tasks and to enhance the model's precision in predicting the existence or nonexistence of vulnerabilities. The Adam optimizer was selected due to its adaptive learning rate modification mechanism, which promoted swift model convergence; the learning rate was established at 0.003, a figure determined from initial trials to guarantee rapid convergence and stability during training. The Sigmoid activation function, suitable for the output layer of probability forecasts, was utilized. In dataset management, 80% of the data was designated as the training set for model development and optimization, whereas the 20% served was considered the test set to assess the model's performance on novel data. The parameter configurations were based on established best practices from the literature and preliminary experimental trials, designed to optimize training efficiency and ensure the validity of the outcomes.

4.1.4 Evaluation Metrics

To thoroughly assess and contrast the methodologies presented in this study, we utilized four commonly employed evaluation measures. The measures include Accuracy, Precision, Recall, and F1 Score.

4.2 Comparative Experiments

4.2.1 Compare Feature Extraction and Model Architectures for Smart Contract Vulnerability Detection

This approach integrates ChatGPT, the specialized Fine-CodeBert, and a BiLSTM network to tackle proficiently key challenges in smart contract vulnerability detection, such as erroneous feature extraction, inadequate model generalization, and inability to manage intricate vulnerability characteristics. Finetuning Fine-CodeBert improves the accuracy of feature extraction and enables it to discern the intricate semantics of Solidity code more effectively, especially in detecting concealed vulnerability patterns. The BiLSTM network supplements the model's comprehension of code sequences via its bidirectional architecture, hence improving its adaptability and generalization across various vulnerability types. Utilizing ChatGPT for code trimming effectively eliminates essential code snippets, minimizes redundancy, and improves detection efficiency for intricate vulnerabilities, such as Ethereum locking and TX issues. The presented approach demonstrates remarkable enhancements in critical measures, including accuracy and recall, relative to conventional deep learning methods.

Comparison studies utilizing the Slither Audited Smart Contracts dataset were performed. In the feature extraction phase, Fine-CodeBert was employed to extract code features, and its efficacy was compared with that of the Word2Vec model trained on Solidity data [32]. Conventional deep learning models, including LSTM, GRU, RNN, and BiLSTM, were utilized in model training for comparative analyses to evaluate the adaptability and efficacy of various neural architectures in vulnerability detection tasks. The findings in Table 4 reveal the substantial superiority of Fine-CodeBert.

Table 4 demonstrates that models utilizing Fine-CodeBert for feature extraction exceed those adopting Word2Vec across all metrics. For example, in the context of RENT vulnerabilities, Fine-CodeBert consistently achieves accuracy rates of 95%, regardless of the application of the BiLSTM, LSTM, RNN, or GRU models, whereas Word2Vec's

maximum accuracy is 89%. In addition, Fine-CodeBert displays average improvements of approximately 8%, 12%, and 12% in accuracy, recall, and F1 scores, respectively, which indicate its superior ability to comprehend intricate semantic information linked to vulnerabilities.

This inclination is similarly observed in the recognition of supplementary vulnerabilities. For instance, in detecting locked Ether vulnerabilities, Fine-CodeBert enables models to obtain accuracy rates ranging from 96% to 99%, which exceed that of Word2Vec by greater than 11% in accuracy and over 16% in F1 score. Models employing Fine-CodeBert for TX vulnerabilities realize accuracy rates of 97% across all neural architectures, and F1 scores consistently reach 95%. By contrast, Word2Vec realizes a maximum accuracy of 92% and an F1 score of 87%.

Table 4. Feature Extraction and Classification Model Cross-Combination Experiment Results.

Vulnerability	Model	Word2Vec				CodeBert			
		ACC	P	R	F1	ACC	P	R	F1
RENT	GRU	89	84	76	80	95	93	90	92
	LSTM	89	84	76	80	95	93	90	92
	RNN	88	82	75	79	95	92	91	92
	BiLSTM	90	85	76	80	**96**	**93**	**91**	**92**
LE	GRU	85	84	68	75	97	96	94	95
	LSTM	85	81	73	77	96	94	93	93
	RNN	84	75	77	76	96	93	94	93
	BiLSTM	86	75	80	77	**99**	**98**	**97**	**98**
TX	GRU	92	89	85	87	98	97	95	96
	LSTM	92	89	83	86	97	95	96	95
	RNN	90	86	83	85	97	95	96	95
	BiLSTM	91	85	87	86	**98**	**96**	**96**	**96**
UR	GRU	92	90	88	89	97	96	94	95
	LSTM	92	90	87	88	97	96	96	96
	RNN	91	89	86	88	97	95	94	95
	BiLSTM	92	90	89	89	**97**	**96**	**96**	**96**

The GCB-DVIA framework skillfully addresses the principal issues in smart contract vulnerability detection—specifically, erroneous feature extraction, insufficient model generalization, and limited efficacy in recognizing intricate vulnerabilities—by incorporating Fine-CodeBert and BiLSTM alongside ChatGPT's code pruning phase. The experimental results clearly establish Fine-CodeBert's capability in identifying hidden vulnerabilities within the code, hence substantially improving model performance in complex vulnerability scenarios and supporting smart contract security.

4.2.2 Comparison with Traditional Tools

To validate the efficacy of the proposed GCB-DVIA method, comparison experiments were performed utilizing the ScrawID dataset against established traditional smart contract security analysis tools, including Slither, Mythril, SmartCheck, Oyente, and Osiris. Table 5 compares and details performance metrics, including accuracy and recall for various vulnerabilities, and thus provides robust data to measure the overall efficacy of

each method. In Table 5, the dash ("-") indicates that the instrument cannot identify certain sorts of vulnerabilities, which underscores the coverage limitations of conventional techniques. The GCB-DVIA framework achieves superior detection capabilities for diverse prevalent vulnerability types, especially in identifying reentrancy vulnerabilities and locked Ether vulnerabilities, and considerably surpasses traditional tools in key metrics of accuracy and recall.

Table 5. Comparison Results of the Proposed Method with Traditional Tools (Vuln-type represents vulnerability type).

Vuln-type	Metric	GCB-DVIA	Slither	Mythril	Smartcheck	Oyente	Osiris
RENT	ACC	**90.69%**	77.86%	79.16%	-	-	44.56%
	R	**84.99%**	60.65%	62.95%	-	-	1.46%
LE	ACC	**95.51%**	92.6%	-	92.5%	-	-
	R	**94.98%**	60.73%	-	59.66%	-	-
TX	ACC	**97.8%**	97.05%	99.6%	97.68%	-	-
	R	**97.52%**	4.94%	88.33%	25.44%	-	-

For RENT vulnerabilities, GCB-DVIA obtained an accuracy of 90.69% and a recall of 84.99%. By contrast, conventional tools such as Slither and Mythril only reached accuracies of 77.86% and 79.16%, respectively, and recall rates of 60.65% and 62.95%, respectively. T his result implies that GCB-DVIA not only detects vulnerabilities with greater precision but also includes a larger share of genuine vulnerability occurrences. The recall inadequacies of conventional tools may lead to a considerable quantity of undetected potential vulnerabilities and fail to meet the stringent security standards of smart contracts.

In identifying locked Ether vulnerabilities, GCB-DVIA demonstrated superior performance and attained accuracy and recall rates of 95.51% and 94.98%, respectively, and remarkably surpassed the efficacy of Slither and SmartCheck. GCB-DVIA considerably improved recall rates by more than 34% and 35% compared with Slither and SmartCheck, respectively. This outcome illustrates GCB-DVIA's capacity to identify locked Ether vulnerabilities more thoroughly, decrease the occurrence of overlooked incidents, and substantially strengthen smart contract security.

In addressing TX vulnerabilities, GCB-DVIA reached an accuracy of 97.8%, which was inferior to Mythril's 99.6%, although it greatly surpassed Mythril in recall and achieved 97.52% versus Mythril's 88.33%. This outcome demonstrates that GCB-DVIA not only maintains high accuracy but also offers a more thorough recognition of vulnerability occurrences, hence minimizing missed detections. The TX vulnerability dataset comprises only 283 positive samples, which causes a considerable data imbalance that could limit a comprehensive assessment of accuracy. Consequently, the more equitable RENT vulnerability dataset is predominantly utilized to highlight the benefits of the GCB-DVIA architecture.

The experimental results unequivocally validate the efficacy and superiority of GCB-DVIA in vulnerability identification, notably its substantial enhancement in recall, a vital parameter, signifying its robust capacity to reliably identify potential weaknesses. The exceptional efficacy of GCB-DVIA in vulnerability identification is due to its rectification of two substantial shortcomings of conventional methods. First, traditional techniques pre-

dominantly rely on established rules and pattern recognition and render the identification of unique or changing vulnerabilities challenging. GCB-DVIA utilizes the self-learning capabilities of deep learning models to enable the automatic extraction of vulnerability features and to adapt to the quick evolution of vulnerabilities. Second, conventional tools are prone to disruption while handling contracts with excessive irrelevant code. GCB-DVIA employs ChatGPT for code pruning and concentrates on the essential components associated with vulnerabilities, thus minimizing extraneous information and improving detection precision and efficacy.

4.3 Ablation Study

In the ablation study, the e ffects of the GPT trimming step and the finetuning of the CodeBert model within the GCB-DVIA framework on overall performance were comprehensively investigated. This paper aims to handle two important issues: The first is the abundance of redundant, irrelevant information in the code that may impede the model's vulnerability detection efficacy. The second is the insufficient feature extraction capacity of pretrained models lacking finetuning in specific domains, which hinders the comprehensive capture of intricate semantic features in smart contract code.

To evaluate the individual effects of each component, two sets of comparative experiments were devised: one eliminating the GPT trimming phase (labeled as No-GPT) and another excluding the finetuning of CodeBert (labeled as No-Fine). This methodology facilitated the assessment of the influence of each component on the model's performance and thereby confirmed the efficacy of the recommended techniques. The results in Table 6 reveal that by integrating GPT pruning with finetuned CodeBert, the GCB-DVIA framework achieved substantial enhancements in the detection of all vulnerability classes.

Table 6. Results of the Ablation Study.

Vulnerability	Metric	No-GPT	No-Fine	GCB-DVIA
RENT	ACC	94%	92%	**96%**
	P	91%	89%	**93%**
	R	91%	85%	**91%**
	F1	91%	86%	**92%**
LE	ACC	96%	96%	**99%**
	P	94%	94%	**98%**
	R	93%	93%	**97%**
	F1	93%	93%	**98%**
TX	ACC	97%	94%	**98%**
	P	95%	93%	**96%**
	R	96%	92%	**96%**
	F1	95%	93%	**96%**
UR	ACC	97%	95%	**97%**
	P	96%	93%	**96%**
	R	96%	94%	**96%**
	F1	96%	94%	**96%**

In relation to RENT vulnerabilities, the accuracy of GCB-DVIA increased from 94% without GPT pruning (No-GPT) to 96%, precision rose from 91% to 93%, and recall and

F1 scores advanced from 91% to 92%. This outcome presents that GPT pruning efficiently removes extraneous code, sharpens the model's emphasis on essential vulnerability characteristics, and strengthens detection precision and thoroughness.

The enhancement in performance for detecting locked Ether vulnerabilities was substantially greater. The accuracy improved by 3% and rose from 93% in the No-GPT setup to 96%. The F1 score greatly increased by 5%, from 93% to 98%. The results indicate that GPT trimming is essential for minimizing code redundancy and focusing on critical vulnerability elements, thereby markedly increasing the model's detection efficacy.

GCB-DVIA achieved superior performance in all evaluation metrics for TX vulnerabilities and UR value vulnerabilities. This result highlights the substantial benefits of finetuning the CodeBert model. The refined CodeBert is more suited to the exact syntax and semantic characteristics of the smart contract domain, which enables it to derive more profound code representations and enhance the model's capacity to identify intricate vulnerability patterns.

The experimental findings unequivocally demonstrate the efficacy of GPT trimming and CodeBert finetuning in improving model performance. This study presents the GCB-DVIA method, which effectively alleviates code redundancy interference and increases feature extraction in classic vulnerability detection methods using GPT pruning and CodeBert finetuning. GPT pruning enables the model to focus on essential vulnerability elements and minimize extraneous information. The refined CodeBert strengthens the model's comprehension of profound semantic characteristics in smart contract code. The consolidation of these two methodologies greatly improves the precision and dependability of smart contract vulnerability identification and underscores their essential function in managing intricate code semantics and improving detection efficacy.

4.4 Generalization Discussion

This study confirms the efficacy of the GCB-DVIA framework utilizing the Solidity language; nonetheless, it is important to recognize that the existing framework has specific constraints when applied to other smart contract languages. These constraints are chiefly evident in the subsequent facets:

Syntax and Semantic Differences: Diverse smart contract languages demonstrate substantial variations in syntax structures and semantic characteristics. The Vyper language eliminates intricate features like inheritance and function overloading, prioritizing simplicity and security. Other blockchain platforms utilize languages such as Michelson for Tezos and Plutus for Cardano, which possess fundamentally distinct programming paradigms and semantic frameworks. The models within the GCB-DVIA framework are trained and fine-tuned on Solidity code, making their straight application to other languages potentially inadequate for accurately capturing their distinct syntax and vulnerability patterns.

Applicability of Pre-trained Models: The framework employs pre-trained models, like CodeBert and GPT, which have been fine-tuned on Solidity to offer specialized feature extraction capabilities. Nonetheless, when analyzing alternative smart contract languages, these models may inadequately extract essential information, resulting in diminished vulnerability detection efficacy. The characteristics specific to a language and its security patterns require the models to be re-tuned for the intended language.

Lack of Multilingual Vulnerability Datasets: The efficacy of the GCB-DVIA architecture depends on the accessibility of a high-caliber vulnerability dataset. At now, there is an absence of publicly accessible vulnerability datasets for other smart contract languages, especially those that are newer or less often utilized. This constraint impedes the training and validation of the framework in these languages, thereby impacting its generalization capacity.

Differences in Vulnerability Types: Various languages may possess distinct vulnerabilities and security issues. Reentrancy vulnerabilities prevalent in Solidity may be absent in other programming languages or may present differently. This necessitates the framework's capacity to identify diverse vulnerability types, while existing models are predominantly trained on prevalent vulnerabilities in Solidity.

To mitigate the previously described constraints and improve the applicability of the GCB-DVIA framework to more smart contract languages, we propose the following measures:

Collection and Construction of Multilingual Vulnerability Datasets: Gather an extensive collection of code samples and documented vulnerability instances in the target language to create a high-quality vulnerability dataset. This will furnish the requisite data support for model training, facilitating the model's acquisition of specific vulnerability patterns in the target language.

Cross-Language Fine-Tuning and Training of Models: Refine CodeBert and GPT models in the target language, or develop new pretrained models. Enhancing the model's feature extraction capability and vulnerability detection performance is achieved by adjusting to the syntactic and semantic characteristics of various languages.

Adjustment of Model Architecture and Detection Algorithms: We adjust the model's input representation and the detection technique based on the attributes of other languages. For instance, based on the grammatical attributes of the target language, the model's encoding method and neural network architecture are modified to successfully identify its distinct susceptibility patterns.

Development of a Modular Framework: Construct a highly extendable modular architecture capable of accommodating new smart contract languages with little alterations. The modular design segregates the language-specific processing flow, enhancing the framework's generalization capability and adaptability.

By recognizing the existing framework's constraints in accommodating alternative smart contract languages and suggesting feasible enhancement options, our study attains more comprehensiveness and equilibrium. This methodology not only augments the practical use of the GCB-DVIA architecture but also offers guidance for subsequent research in the domain of smart contract security detection.

5. CONCLUSION

5.1 Summary of Research

This research presents a methodology for detecting vulnerabilities in intelligent contracts, utilizing big language models and deep learning, termed GCB-DVIA. The approach initially utilizes ChatGPT for code trimming, subsequently employs a fine-tuned

CodeBert model for feature extraction, and ultimately applies a bidirectional long short-term memory network integrated with a self-attention mechanism for vulnerability identification and categorization. The experiments detailed in the paper were confirmed using two public datasets of Ethereum smart contracts, yielding significant results. The detection accuracy for RENT vulnerabilities attained 96%, while for TX vulnerabilities, it achieved 98%. The GCB-DVIA architecture surpassed standard methods in multiple critical performance parameters, particularly in recall and F1 score. Furthermore, studies revealed that employing the Fine-CodeBert model for feature extraction enhanced detection accuracy by 5%-10% relative to the Word2Vec model.

5.2 Future Work

Future work primarily concentrates on two facets: advancing vulnerability detection methodologies, particularly for intricate vulnerabilities prevalent in DeFi platforms, such as flash loan attacks; and exploring Few-shot techniques or self-supervised learning for model training on limited datasets.

REFERENCES

1. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
2. M. Di Angelo and G. Salzer, "Consolidation of ground truth sets for weakness detection in smart contracts," in *International Conference on Financial Cryptography and Data Security*. Springer, 2023, pp. 439–455.
3. C. Jentzsch, "Decentralized autonomous organization to automate governance," *White paper, November*, 2016.
4. X. Tang, K. Zhou, J. Cheng, H. Li, and Y. Yuan, "The vulnerabilities in smart contracts: A survey," in *Advances in Artificial Intelligence and Security: 7th International Conference, ICAIS 2021, Dublin, Ireland, July 19-23, 2021, Proceedings, Part III 7*. Springer, 2021, pp. 177–190.
5. A. Ghaleb and K. Pattabiraman, "How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 415–427.
6. Y. Sun and Y. Lee, "Embedding-based two-stage entity alignment for cross-lingual knowledge graphs." *Journal of Information Science & Engineering*, Vol. 40, no. 2, 2024.
7. G. Grieco, W. Song, A. Cygan, J. Feist, and A. Groce, "Echidna: effective, usable, and fast fuzzing for smart contracts," in *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*, 2020, pp. 557–560.
8. S. So, M. Lee, J. Park, H. Lee, and H. Oh, "Verismart: A highly precise safety verifier for ethereum smart contracts," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1678–1694.
9. B. Mueller, "Mythril-reversing and bug hunting framework for the ethereum blockchain," 2017.
10. J. Huang, K. Zhou, A. Xiong, and D. Li, "Smart contract vulnerability detection model based on multi-task learning," *Sensors*, Vol. 22, no. 5, 2022, p. 1829.

11. Z. Liu, M. Jiang, S. Zhang, J. Zhang, and Y. Liu, "A smart contract vulnerability detection mechanism based on deep learning and expert rules," *IEEE Access*, 2023.
12. C. Sendner, H. Chen, H. Fereidooni, L. Petzi, J. König, J. Stang, A. Dmitrienko, A.-R. Sadeghi, and F. Koushanfar, "Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning." in *NDSS*, 2023.
13. 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, 2023, p. 103423.
14. 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, 2020, pp. 1133–1144.
15. Y. Wu, D. Zou, S. Dou, S. Yang, W. Yang, F. Cheng, H. Liang, and H. Jin, "Scdetector: Software functional clone detection based on semantic tokens analysis," in *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, 2020, pp. 821–833.
16. S. Badruddoja, R. Dantu, Y. He, M. Thompson, A. Salau, and K. Upadhyay, "Smarter contracts to predict using deep-learning algorithms," in *2022 Fourth International Conference on Blockchain Computing and Applications (BCCA)*. IEEE, 2022, pp. 280–288.
17. X. Tang, Y. Du, A. Lai, Z. Zhang, and L. Shi, "Lightning cat: A deep learning-based solution for smart contracts vulnerability detection," 2023.
18. Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, "When gpt meets program analysis: Towards intelligent detection of smart contract logic vulnerabilities in gptscan," *arXiv preprint arXiv:2308.03314*, 2023.
19. S. Hu, T. Huang, F. İlhan, S. F. Tekin, and L. Liu, "Large language model-powered smart contract vulnerability detection: New perspectives," *arXiv preprint arXiv:2310.01152*, 2023.
20. W. Lin and L.-C. Yu, "Scarce resource dimensional sentiment analysis using domain-distilled bert." *Journal of Information Science & Engineering*, Vol. 39, no. 2, 2023.
21. R. Ahuja and S. Sharma, "B^2grua: Bertweet bi-directional gated recurrent unit with attention model for sarcasm detection." *Journal of Information Science & Engineering*, Vol. 39, no. 4, 2023.
22. J. Feist, G. Grieco, and A. Groce, "Slither: a static analysis framework for smart contracts," in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 2019, pp. 8–15.
23. P. U. HEPSA, S. A. OZEL, and A. YAZICI, "Transfer learning with fuzzy for breast cancer," *Journal of Information Science and Engineering*, Vol. 40, 2024, pp. 919–939.
24. J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.
25. Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

26. H. Zhang, W. Zhang, Y. Feng, and Y. Liu, "Svscanner: Detecting smart contract vulnerabilities via deep semantic extraction," *Journal of Information Security and Applications*, Vol. 75, 2023, p. 103484.
27. J. Li, G. Lu, Y. Gao, and F. Gao, "A smart contract vulnerability detection method based on multimodal feature fusion and deep learning," *Mathematics*, Vol. 11, no. 23, 2023, p. 4823.
28. V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, Vol. 1, 2013, pp. 22–23.
29. "Smart contract weakness classification and test cases," Available online, accessed: 2024-04-12. [Online]. Available: http://swcregistry.io
30. C. S. Yashavant, S. Kumar, and A. Karkare, "Scrawld: A dataset of real world ethereum smart contracts labelled with vulnerabilities," *arXiv preprint arXiv:2202.11409*, 2022.
31. M. Rossini, "Slither audited smart contracts dataset," 2022.
32. K. W. Church, "Word2vec," *Natural Language Engineering*, Vol. 23, no. 1, 2017, pp. 155–162.

Yunhao Zhao is a postgraduate student at the School of Information Science and Engineering, Linyi University. He focuses on artificial intelligence and information security. He can be reached at 220854042002@lyu.edu.com.

Hao Zhang is a master's student at the School of Information Science and Engineering, Linyi University. He specializes in artificial intelligence and data science, with a particular emphasis on medical image segmentation. He can be contacted via haozhang14688@163.com.

Lianjie Wang is a master's student at the School ofInformation Science and Engineering, Linyi University.He specializes in Internet of Things and Information Security, with a particular emphasis on Blockchain technology. He can be contacted via wanglianjie406@outlook.com.

Keqing Wang is a master's student in the School of Information Science and Engineering at Linyi University.She focuses on in-depth study of relevant theories and technologies in the field of computer science, and is mainly proficient in the direction of blockchain technology.She can be contacted via 18366753862@163.com.

Wanshui Song received the M.D. degree in School of Information Science and Engineering from Shandong Normal University, China in 2022. He is currently studying toward a Ph.D. degree in the computer science and technology from Harbin Engineering University, China. His current research interests include blockchain, network science, data mining.

Zhongwen Zhang graduated from the Institute of Information Engineering, Chinese Academy of Sciences. She has held senior engineering and security expert roles at Huawei Technologies Co., Ltd. and Ant Group, and has a deep professional background in blockchain technology. Currently, Dr. Zhang is employed at Linyi University, focusing on research in blockchain security. She is not only an expert in blockchain technology but also a lecturer at the School of Information Science and Engineering at Linyi University. Dr. Zhang is committed to advancing the security research of blockchain technology and serves as a bridge between academia and the industry.

Wenyin Zhang is currently a professor at the School of Information Science and Engineering, Linyi University. He obtained master's and doctor's degrees in computer science and technology from Shandong University of Science and Technology and Chinese Academy of Sciences respectively. Her research interests include image processing and information security. He has published over 50 papers and has been included in SCI or EI.

Yeh-Cheng Chen is currently pursuing the Ph.D. degree with the Department of Computer Science, University of California, Davis, CA, USA. His research interests include radio-frequency identication (RFID), data mining, social networks, information systems, wireless network articial intelligence, the IoT and security.