Enhancing Reentrancy Vulnerability Detection and Repair with a Hybrid Model Framework

Mengliang Li[†§], Xiaoxue Ren[*†§], Han Fu[¶], Zhuo Li[‡], Jianling Sun[¶]
[†*]*State Key Laboratory of Blockchain and Data Security, Zhejiang University,* [‡]*State Street Technology(Zhejiang) Ltd*
[¶]College of Computer Science and Technology, Zhejiang University
[§]Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security
123lmliang@zju.edu.cn,xxren@zju.edu.cn,11821003@zju.edu.cn,lizhuo@zju.edu.cn,sunjl@zju.edu.cn

***Abstract*—Smart contracts bring revolutionary changes to the credit landscape. However, their security remains intensely scrutinized due to numerous hacking incidents and inherent logical challenges. One well-known and representative issue is reentrancy vulnerability, exemplified by DAO attacks that lead to substantial economic losses. Conventional approaches to detecting and repairing reentrancy vulnerability often suffer from numerous limitations, including disregarding the intricate vulnerability features and the overfitting problems associated with imbalanced datasets. Large language models are distinguished for their excellent language understanding and have achieved explosive success in artificial intelligence. However, direct prompt-based LLMs-driven approaches for reentrancy vulnerability are plagued by inefficiencies and a lack of domain-specific vulnerability knowledge. This paper proposes a hybrid framework to enhance reentrancy vulnerability detection and repair and safeguard smart contract security. This unified framework comprises two crucial modules: enhanced DL-driven vulnerability detection and knowledge-aware LLMs-driven vulnerability repair. Our approach can significantly enhance reentrancy vulnerability detection and repair efficiency by integrating advanced techniques such as feature extraction, data balancing, deep learning networks, and knowledge-aware prompting. Extensive experimental results validate the superiority of our approach over state-of-the-art baselines, emphasizing its potential to fortify the security of smart contracts and blockchain-based systems. For instance, our approach can achieve 3.51%, 2.31%, 0.42%, and 0.85% improvements in accuracy, recall, precision, and F1 score while detecting reentrancy vulnerability. Additionally, our approach also can achieve a 9.62% improvement in reentrancy vulnerability repair.**

***Index Terms*—Smart Contract, Vulnerability Detection and Repair, LLMs, Reentrancy**

I. Introduction

The smart contract is a program that automatically executes within a blockchain system, managing data and assets within that system [1]. As the use of smart contracts proliferates across various fields, their security remains a contentious issue due to numerous hacking incidents and inherent logical challenges [2]. For instance, a vulnerability in a smart contract, such as the one exploited in the DAO attack, has resulted in financial losses amounting to $150 million [3]. Smart contracts exhibit sophisticated types and a significant number of vulnerabilities. The SWC registry summarizes 37 types of vulnerabilities, while the NCC Group identifies the top 10 vulnerabilities [3]. Among these, reentrancy is the most notorious and representative [4], the root cause of the DAO attacks on smart contracts. Therefore, detecting and repairing reentrancy vulnerabilities is crucial for advancing the overall security of smart contracts.

Existing approaches for detecting and repairing reentrancy vulnerability can be categorized into three types: rule-based approaches [5]–[7], deep learning-based approaches [8], and large language models-based (LLMs-based) approaches [9]–[12]. The representative tools in all fields, include Oyente [13], Slither [7], Securify [14], SmartCheck [15], SliSE [16], Mythril [17], Rechecker [18], EVMPatch [19], TMP [20], AME [21], CGE [22]. For instance, Zhuang [20] proposed a graph neural network-based (GNN-based) for detecting various smart contract vulnerabilities, which transforms basic functions, relevant variables, and callback functions into key node features. Liu [21] introduced a smart contract vulnerability detection method that integrates expert knowledge with graph neural networks, enabling the extraction of both global graph vulnerability features and local expert features.

However, several challenges remain despite the significant work proposed to detect and repair reentrancy vulnerabilities. These include: (1) **Inadequate Features:** The conventional approach heavily relies on pre-defined expert rules, which neglect the deeper vulnerability features and sophisticated function call chains in reentrancy [5]. For instance, previous work [20], [21] focuses primarily on the attack features of smart contract vulnerabilities, while ignoring the functional dependency features of these vulnerabilities. (2) **Imbalanced Datasets:** The conventional approach utilizes the imbalanced dataset to train the deep learning network, leading to the overfitting problem and hurt experiment performance [23]–[25]. Therefore, it's crucial to tackle the imbalanced dataset problem before training vulnerability features. (3) **Inadequate Domain-Specific Knowledge:** The contemporary approach integrates ChatGPT with a simplistic detection prompts [9]–[12] without the domain-specific knowledge of reentrancy vulnerability and adequate structural analysis of smart contracts.

Approach: In this paper, we propose a unified framework to detect and repair reentrancy vulnerabilities by addressing the aforementioned challenges. This framework encompasses two crucial phases: enhanced deep learning-based vulnerability detection and knowledge-aware prompt-based vulnerability

[*]Corresponding Author: Xiaoxue Ren

repair. (1) **Vulnerability Detection Phase:** In the detection phase, we introduce a novel feature extraction algorithm capable of capturing the key characteristics of reentrancy vulnerabilities and inter-function dependency features. Additionally, we propose a novel algorithm to augment imbalanced datasets while maintaining the semantic and syntactic integrity of code gadgets. These two algorithms address the challenges faced by conventional deep learning-based tools, such as inadequate features and imbalanced datasets. (2)**Vulnerability Repair Phase:** In the repair phase, we propose a knowledge-aware dynamic prompt-based approach to enhance LLMs' ability to repair reentrancy vulnerabilities. This prompt-based approach is constructed with examples of vulnerabilities, fixed examples, repair approaches, and deep learning attention score-based code slices. This approach enables LLMs to autonomously learn how to fix the input code gadget with pre-domain reentrancy knowledge. Incorporating deep learning attention score-based code slices in the prompts aims to leverage intermediate states from the deep learning model's training process, aiding the LLMs in more accurately locating vulnerabilities.

Extensive experiments have demonstrated that our method enhances reentrancy vulnerabilities' detection and repair capabilities, showcasing its immense potential in safeguarding smart contract security. (1) **Performance Improvement:** Our method surpasses other advanced benchmark methods, including rule-based, DL-based, and LLMs-based approaches. It improves the accuracy, recall, precision, and F1-score of vulnerability detection by 3.51%, 2.31%, 0.42%, and 0.85%, respectively. Furthermore, our method achieves a 9.62% increase in the repair rate. (2)**Ablation Study:** Comparative experiments on feature extraction, data balancing, and cross-language pre-trained models demonstrate that our method effectively addresses issues such as insufficient vulnerability features, data imbalance, and inadequate feature representation. Furthermore, we summarize three repair approaches and discuss the impact of the order of examples in constructing effective knowledge-aware prompts. This guidance aids researchers in better crafting repair prompts. Notably, this is the first work to combine LLMs with knowledge-aware prompts for repairing smart contract reentrancy vulnerabilities.

We summarize our contributions as follows:

1) **Hybrid Model Framework:** We propose a hybrid model-driven framework to detect and repair reentrancy. This is the first work to integrate the enhanced deep learning-based approach with the knowledge-aware prompt-based approach for reentrancy study.
2) **Enhanced Detection Phase:** In the vulnerability detection phase, we introduce a multi-function-based feature extraction algorithm, a data augmentation algorithm, and a cross-language representation-based deep neural network. These advancements address challenges such as inadequate features, data imbalance, and insufficient code representation during deep-learning model training.
3) **Enhanced Repair Phase:** In the vulnerability repair phase, we propose a knowledge-aware prompt-based approach to repair reentrancy vulnerability. This method leverages the deep learning attention score-based code slices and pre-domain vulnerability repair knowledge, significantly enhancing the LLMs' ability to locate and repair vulnerabilities.

II. Related Work

A. Smart Contract Vulnerability

Smart contracts are immutable and cannot be modified once deployed, a feature that significantly distinguishes them from programs written in other languages [1]. Despite this immutability, smart contract security is still under great controversy due to frequent hacker attacks and logical vulnerabilities. Recent studies on smart contract security can be categorized into three distinct layers based on where vulnerabilities occur [26]–[28]: (1) Solidity Layer: Vulnerabilities at the solidity layer arise from logical errors in smart contract programming language. Notable examples in this category include reentrancy, integer overflow, access control, and exception handling. (2) EVM Layer: Vulnerabilities at the Ethereum virtual machine (EVM) layer stem from the special requirements and characteristics of the EVM. The representative vulnerabilities include the short address, TX.origin, and ethereum loss. (3) Blockchain Layer: Vulnerabilities at the blockchain layer occur during the inter-node communication within the blockchain network. The significant vulnerabilities in these categories include timestamp dependency, order dependency, and transaction dependency.

B. Detection and Repair

Previous studies for detecting and repairing reentrancy vulnerability can be categorized into the following categories: rule-based [5]–[7], [13], deep learning-based [8], [20]–[22], [29], [30], and LLMs-based [9]–[12]. The representative tools encompass Oyente [13], Slither [7], Securify [14], SmartCheck [15], SliSE [16], Mythril [17], EVMPatch [19], SMARTSHIELD [31], Rechecker [18], TMP [20], AME [21], CGE [22], DeeSCVHunter [32], and DMT [33]. For instance, Zhuang [20] proposed a graph neural network-based (GNN-based) for detecting three smart contract vulnerabilities, which transforms basic functions, relevant variables, and callback functions into key node features. Liu [21] introduced a smart contract vulnerability detection method that integrates expert knowledge with graph neural networks, enabling the extraction of both global graph vulnerability features and local expert features. Yu [34] proposes a gas-awareness approach to repair smart contract vulnerabilities according to the gas consumption of code patches.

C. LLMs-Driven Smart Contract

ChatGPT, developed by OpenAI [35], gained significant attention in 2022 for its advanced text generation capabilities [36]. As large language models (LLMs) increasingly influence AI, researchers have begun applying these models to smart contract security [9], [12], [37], [38]. For instance, Chen utilized prompt-based ChatGPT and GPT-4 to detect vulnerabilities in the SmartBugs dataset. Although LLMs

show high recall in identifying vulnerabilities, their precision remains limited. The primary causes of false positives include biases in protected mechanisms, misinterpretation of code semantics, distractions from comments, and interference from unexecuted code.

III. Challenge and Motivation

Previous studies have witnessed some achievements in reentrancy vulnerability detection and repair [20], [21], [39]. However, several challenges remain to construct a unified framework for reentrancy vulnerability detection and repair.

Challenge 1: Inadequate Feature The conventional approach to detect and repair reentrancy vulnerability with pre-defined expert rules. This approach requires the researcher to construct a cumbersome algorithm [34], which only focuses on vulnerability features and overlooks the deeper feature relationship. For instance, some works [5] have demonstrated that the sophisticated function-call-chain in smart contracts can significantly affect vulnerability detection and repair.

Challenge 2: Imbalanced Dataset Previous studies utilize imbalanced datasets to train deep learning networks, which can lead to overfitting problems and damage the performance and stability [23]–[25]. This motivates us to take into consideration constructing balanced datasets before the reentrancy vulnerability feature training.

Challenge 3: Inadequate Domain-Specific Knowledge The conventional approach integrates LLMs with simplistic prompts lacking domain-specific vulnerability knowledge, leading to unsatisfactory performance. However, extensive work indicates that an adequate prompt with pre-defined knowledge can facilitate LLMs' understanding of input tasks. These motivate us to contract an adequate knowledge-aware prompt-based approach for LLMs' detection and repair.

Our Motivation: The challenges above motivate us to create a unified framework to enhance the detection and repair of reentrancy vulnerability. To address these challenges, we propose the following approaches:

1) **Approach for challenge 1:** We first propose a feature extraction algorithm capable of capturing the key characteristics of reentrancy vulnerabilities and inter-function dependency features.
2) **Approach for challenge 2:** We employ a novel algorithm to augment imbalanced datasets, mitigating the overfitting problem in the training phase. This approach ensures the preservation of both the semantic and syntactic integrity of code gadgets.
3) **Approach for challenge 3:** We present a knowledge-aware prompt-based approach for vulnerability repair, leveraging the adequate reentrancy vulnerability knowledge to enhance the effectiveness of the vulnerability repair.

IV. Methodology

Figure 1 illustrates the overall framework of our hybrid model-driven approach. This unified framework encompasses two crucial phases: enhanced DL-based vulnerability detection and knowledge-aware prompt-based vulnerability repair. (1) In the detection phase, we first perform a root cause analysis and then introduce a novel feature extraction algorithm alongside a data balancing algorithm. (2) In the repair phase, we extract the attention-based code slices from the DL-based detection result. Then, we construct the knowledge-aware prompt to repair reentrancy vulnerability with the domain-specific reentrancy vulnerability knowledge and attention-based code slices.

A. Detection Phase

1) Root Cause Analysis: The root causes of reentrancy vulnerabilities are associated with DAO attacks [40], a type of external hacker attack targeting the logic of smart contracts. Three conditions must be met for a reentrancy vulnerability to occur: (1) The attacked smart contract must include the *call.value* keyword, which is essential for reentrancy vulnerabilities. (2) To exploit the vulnerability, The external attacker must directly or indirectly invoke the function containing the *call.value* keyword [5]. (3) The external attacker must construct a fallback function to call the function containing the *call.value* keyword. Thus, identifying the code gadgets with the *call.value* keyword and function-call relationship enable the extraction of reentrancy vulnerability features.

2) Feature Extraction: In this section, we propose a new algorithm to extract code gadgets containing vulnerability features and function-call features of reentrancy. We first introduce two definitions and then thoroughly analyze the vulnerability extraction algorithm.

Definition 1: (File, Function, Line, Token) The smart contract dataset contains multiple files $F = \{f_1, f_2, f_3, \ldots, f_i\}$. Each smart contract source code contains multiple functions $Fun = \{fun_1, fun_2, fun_3, \ldots, fun_j\}$. Each function contains multiple lines of code, and each line contains multiple tokens.

Definition 2: (Core Function, Forward Function, Backward Function) A core function contains the keyword *call.value* associated with reentrancy vulnerabilities. The forward function calls the core function, while the core function calls the backward function. Extracting these functions is to obtain features related to reentrancy vulnerabilities and the calling relationships between functions.

Algorithm: Algorithm 1 details how to extract multi-function code slices containing vulnerability features from smart contract source code. The input is a source code dataset containing multiple smart contracts, and the output is a set of smart contract code slices containing vulnerability features. (1) First, we define and initialize the variables for the entire function. In line 1, we define the total code slice output set Out as empty. In line 3, we define the multi-feature function set o_i for each smart contract as empty, the function set fun_{yes} containing reentrancy vulnerability keywords as empty, the function set fun_{no} not containing reentrancy vulnerability keywords as empty, the forward function set $fun_{forward}$ as empty, and the backward function set $fun_{backward}$ as empty. (2) In line 6, we use a function slicing algorithm to obtain the set of all functions Fun for each smart contract. (3) In lines

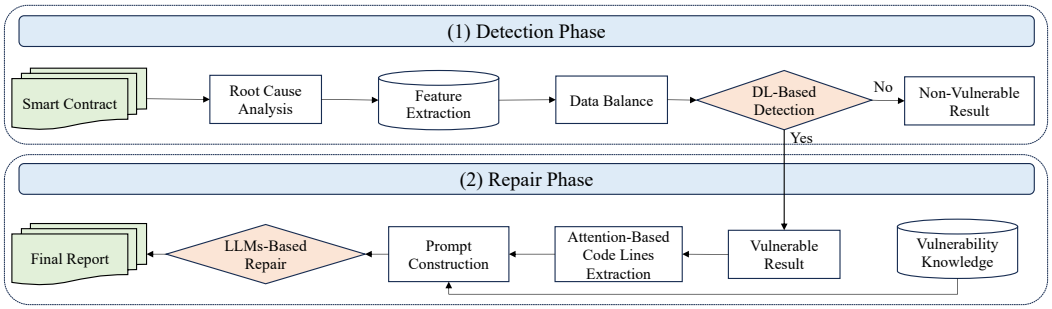

Fig. 1. The overall framework of our approach with two crucial phases: vulnerability detection and vulnerability repair phase.

7-15, we extract the function set fun_{yes} containing vulnerabilities and the function set fun_{no} not containing vulnerabilities from the function set Fun based on the presence of keywords. (4) In lines 16-24, we extract the forward function set $fun_{forward}$ and the backward function set $fun_{backward}$ from the function set fun_{no} based on function call relationships. (5) In line 25, we add the functions $fun_{forward}$ and $fun_{backward}$ corresponding to a smart contract to the function set O_i. In line 26, we add each function set O_i to the total code slice output set Out. Finally, the entire function outputs the total code slice output set Out.

Illustration: Figure 2 shows an example of extracting vulnerabilities from smart contract source code. The smart contract *MultiWallet* contains three functions: *isConfirmed* (lines 4-12), *executeTransaction* (lines 13-24), and *confirmTransaction* (lines 25-31). The function *executeTransaction* contains the *call.value* keyword, and there are function call relationships between *executeTransaction*, *isConfirmed*, and *confirmTransaction*. (1) First, we split the source code into functions. (2) Then, we extract the core function *executeTransaction* based on the presence of the reentrancy vulnerability keyword. (3) Next, we extract the forward function *confirmTransaction*, and the backward function *isConfirmed* based on the call relationships among the three functions. (4) Finally, we output the total code slice set composed of the three function slices as input samples for the neural network.

3) Data Balance: We should balance the extracted code slice dataset before feeding it into the deep learning network due to the imbalanced dataset associated with the overfitting problem. In this section, we propose a novel data balance algorithm to enhance the dataset from the token-based level and sentence-based level.

Illustration: Figure 3 details two approaches for augmenting the imbalanced datasets. Sentence-level insertion involves randomly selecting a line of code from a smart contract code slice and inserting it at the end of the code slice. For instance, in Figure 3, we randomly select the fourth line of code from the function *withdraw* and place it at the end of the entire code slice. Token-level insertion involves randomly selecting a token and inserting it at the beginning or end of the code slice, without changing the overall logical structure. For instance, in Figure 3, we randomly select the token from the function *withdraw* and insert it at the end of the code slice. Overall, these two approaches allow us to enhance an imbalanced dataset while preserving the semantic and syntactic structure of the code slices.

Algorithm 1 Extracting multi-function code gadget of reentrancy vulnerability from smart contract source code.

Input: Smart contract set $F = (f_1, f_2, f_3, ..., f_i)$
Parameter: Function set Fun, function set F_{yes} with keyword (call.value), function set F_{no} without keyword, keyword set K
Output: Code gadget set $Out = (O_1, O_2, O_3 ... O_i)$

1: $Out \leftarrow \emptyset$
2: **for** each file $f_i \in FUN$ **do**
3: $O_i \leftarrow \emptyset$
4: $F_{yes} \leftarrow \emptyset$, $F_{no} \leftarrow \emptyset$
5: $F_{forward} \leftarrow \emptyset$, $F_{backward} \leftarrow \emptyset$
6: Leveraging segmentation algorithm of extracting function to generate function set Fun from f_i
7: **for** each function $fun_j \in Fun$ **do**
8: **for** each keyword $k \in K$ **do**
9: **if** fun_j match k **then**
10: $F_{yes} \leftarrow F_{yes} \cup fun_j$
11: **else**
12: $F_{no} \leftarrow F_{no} \cup fun_j$
13: **end if**
14: **end for**
15: **end for**
16: **for** each function $fun_j \in F_y$ **do**
17: **for** each function $fun_k \in F_n$ **do**
18: **if** function fun_k invoke fun_j **then**
19: $F_{forward} \leftarrow F_{forward} \cup fun_k$
20: **else**
21: $F_{backward} \leftarrow F_{backward} \cup fun_k$
22: **end if**
23: **end for**
24: **end for**
25: Adding the function sets F_{yes}, $F_{forward}$ and $F_{backward}$ into O_i
26: $O_{out} \leftarrow O_{out} \cup O_i$
27: **end for**
28: **return** Out;{the set of multi-function code gadgets}

4) DL-Based Detection: We propose a deep learning-based (DL-based) neural network to train the enhanced code slice dataset. We draw inspiration from pre-trained models, transformer encoders, and convolutional neural networks (CNN) to design the neural network in figure 4.

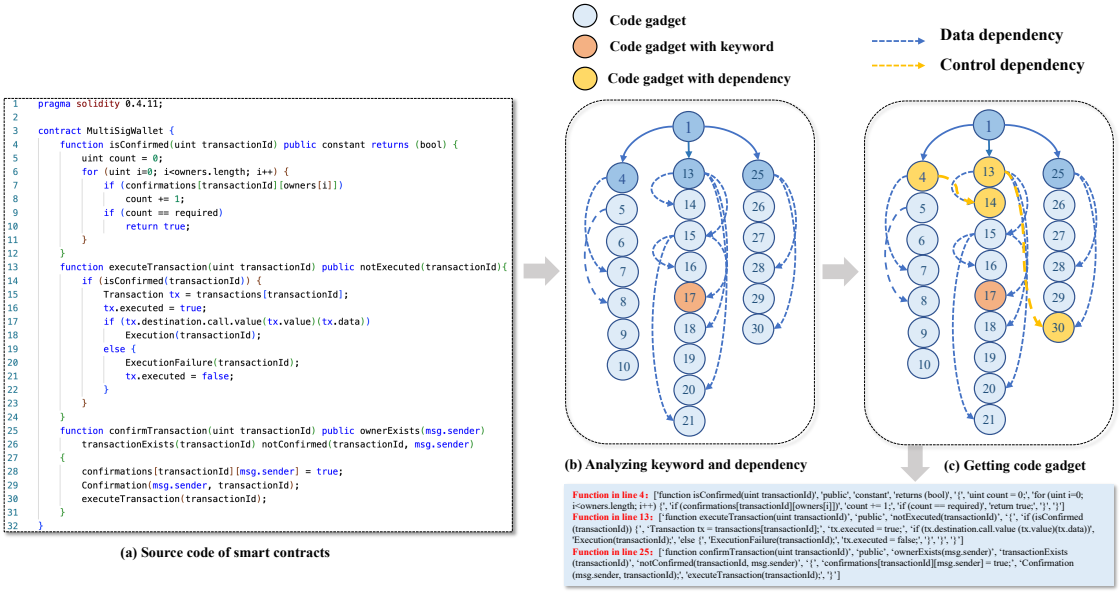

Fig. 2. The illustrative example to extract function-call-based code gadgets.

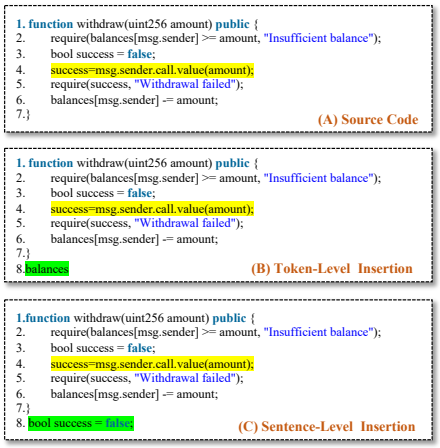

Fig. 3. The two approaches to augment the imbalanced datasets from sentence-level and token-level.

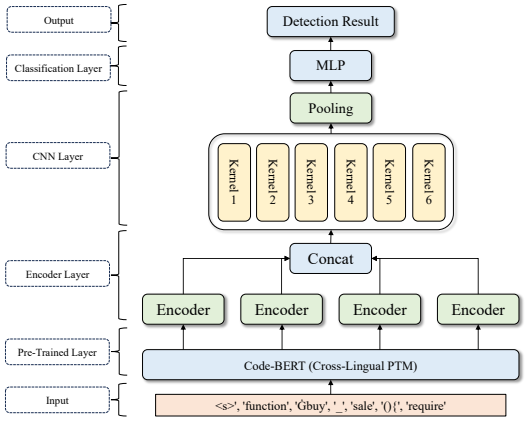

Fig. 4. The overall structure of deep learning network.

These deep learning neural networks include four parts: vulnerability code representation based on a cross-language pre-trained layer, global vulnerability feature extraction based on encoder layers, local feature extraction based on CNN layers, and a classification layer based on MLP. (1) **Pre-Trained Layer:** We utilize cross-language pre-trained code representation models (CodePTM, Code Pre-Trained Models) to vectorize the Solidity language of smart contracts. Since no pre-trained models specifically for Solidity exist, we choose pre-trained models trained in other languages to represent smart contract code slices. Pre-trained code representation models in our approach include CodeBERT, Roberta, and CodeBERTa. (2) **Encoder Layer:** We utilize transformer encoders to extract global vulnerability features of reentrancy vulnerabilities. The core principle of the encoder is the self-attention mechanism. This mechanism first transforms the input vector into Q, K, and V, then calculates the input vector's self-attention weights. (3) **CNN Layer:** We employ the convolution kernels to extract local features of reentrancy vulnerability code slices. The CNN layer can extract the local features of code slices. After using convolution kernels to extract multiple features of code slices, we use pooling techniques to reduce dimensions, aiming to extract the most important features. In our study, we utilized 6 convolution kernels of size 2 and AveragePooling to obtain the average features from the convolution kernels. (4) **Classification Layer:** After extracting code slice features using convolution kernels and pooling techniques, we feed them into a multi-layer perceptron (MLP, see classification layer in Figure 4) for classification training, aiming to identify the presence of reentrancy vulnerabilities in the code. We utilize the softmax as the activation function and Cross-Entropy as the loss function.

B. Repair Phase

After obtaining the code gadget from the deep-learning neural network, we propose a knowledge-aware prompt-based approach to repair smart contract reentrancy vulnerabilities. We first introduce the workflow of obtaining the attention-based code lines and then construct the knowledge-aware prompt with domain-specific vulnerability knowledge to LLMs.

1) Attention-Based Code Lines Extraction: The deep learning network assigns weights to tokens in the input code slices during training. Therefore, we extract code lines with high attention scores as location prompts. Figure 5 illustrates an example of extracting attention scores from code lines. We first extract the score distribution of all tokens in a code slice from the convolutional layer of the neural network (where yellow lines have the highest attention scores, gray lines have the next highest, and unmarked lines have the lowest). We utilize the high-attention yellow lines as inputs for the LLMs to help them precisely locate the vulnerabilities.

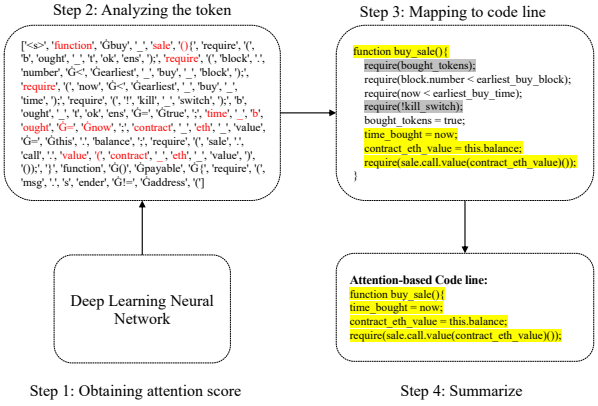

Fig. 5. The workflow of obtaining attention score-based code lines.

2) Prompt Construction: The knowledge-aware prompt-based vulnerability repair consists of four main parts (see repair prompt in figure 6): **vulnerability examples**, **repaired vulnerability examples**, **repair approaches**, and **attention scores-based code lines**. (1) **Vulnerability Example:** The function-level code slice contains a reentrancy vulnerability. (2) **Repaired Example:** The function-level code slice has been repaired using three various methods. (3) **Three Repair Approaches:** These include locking the *call.value* code line, replacing the *call.value* function with *transfer* or *send* functions, and using the check-effects-interactions pattern to ensure all parameters are properly checked before modifying the contract state. (4) **Attention Score-Based Code lines:** We obtain the attention score-based code lines and construct it into repair prompt. This approach can help LLMs better locate the reentrancy vulnerability.

3) LLMs-Based Repair: To evaluate the effectiveness of large language models in repairing reentrancy vulnerabilities, we follow the structured methodology. Initially, we provide the LLMs with a code gadget containing reentrancy vulnerability, accompanied by a knowledge-aware repair prompt. We then

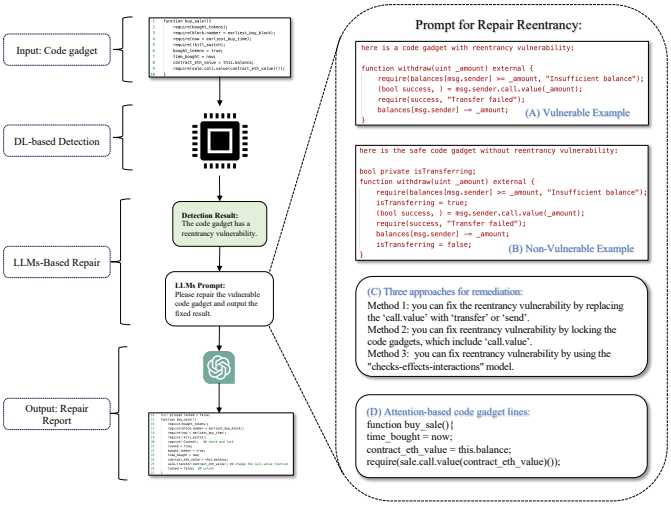

Fig. 6. The workflow of repairing reentrancy vulnerability.

assess the LLMs' ability to produce the repaired version of the code. Following this, we rigorously verify the correctness of the repaired code. Finally, we summarize the LLMs' performance in addressing reentrancy vulnerabilities by calculating the repair success rate and accuracy of the corrected outputs.

V. Experiments

In this section, we first explain the experimental procedures, encompassing datasets and benchmarks, experiment setup, and evaluation metrics. Then, we analyze the experimental results by answering the following three questions to assess the effectiveness of our approaches:

RQ 1: How does our approach perform compared with the existing state-of-the-art methods?

RQ 2: How does our feature extraction data balance algorithm perform in the vulnerability detection phase?

RQ 3: How does our knowledge-aware LLMs-based prompt perform in the vulnerability repair phase?

A. Experiment Preparation

Datasets and Benchmark: We utilize the ESC [18], [21], [33], SmartBugs [17], and ReentrancyStudy [3] datasets, which contain smart contract source codes from Ethereum. We select three types of tools for model comparison: rule-driven tools, including Securify [14], Smartcheck [15], Mythril [17], Oyente [13], Slither [7], Osiris [41], SmartFix [42], sGuard [43], and Elysium [44]; deep learning-driven tools, including MLP, GNN, RNN, TMP [20], AME [21], CGE [22], and ReVulDL [4]; and large model-driven tools, including ChatGLM [45], ChatGPT [46], and CodeLlama [47].

Experiment Setup: We utilize an Alibaba Cloud server as an experimental environment, equipped with an Intel(R) Xeon(R) Platinum 8269CY CPU and 15GB of memory. We utilize TensorFlow to build the deep-learning neural network. We split the training, validation, and test sets in an 8:1:1 ratio. Additionally, we set the learning rate to 0.002, dropout to 0.5, and batch size to 64.

TABLE I
THE OVERALL COMPARISON OF SMART CONTRACT REENTRANCY VULNERABILITY DETECTION.

Method		Accuracy(%)	Recall(%)	Precision(%)	F1 score(%)
LLMs-Based	ChatGPT	50.96	52.94	17.31	26.09
	Codellama-7B	30.74	31.61	18.06	22.99
	Codellama-13B	36.6	34.19	24.53	28.56
	ChatGLM	42.24	38.15	20.06	26.29
Rule-Based	Securify	72.89	73.06	68.4	70.41
	Smartcheck	52	24.32	31.03	27.07
	Osiris	56.73	63.88	40.94	49.90
	Mythril	60	39.21	68.96	50
	Oyente	71.5	50.84	51.72	51.28
	Slither	74.02	73.50	74.44	73.97
	SliSE	-	96.77	68.18	80.00
DL-Based	MLP	78.39	75.66	80.06	77.77
	RNN	79.81	74.85	82.92	78.56
	GCN	77.85	78.79	70.02	74.15
	TMP	84.48	82.63	74.46	78.11
	AME	90.19	89.69	74.06	87.94
	CGE	89.15	87.62	85.24	86.41
	ReVulDL	-	93	92	93
Our approach	HM-RVDR	93.7(3.51↑)	95.31(2.31↑)	92.42(0.42↑)	93.85(0.85↑)

Evaluation Metrics: We evaluate detection performance utilizing accuracy, recall, precision, and F1-score. For repair performance, we use fixed rates and correction rates. The fixed-rate measures the proportion of vulnerabilities the LLMs can attempt to repair, while the correction rate reflects the proportion of successfully repaired vulnerabilities. Both rates are calculated as the ratio of repaired or successfully repaired code slices to the total number of slices.

B. Experiment Result

1) Answer to RQ1 (Overall Comaprision): In this section, we present a comprehensive comparison and analysis of our approach to detecting and repairing reentrancy vulnerabilities, benchmarking it against established methods. For the detection phase, we evaluate our approach alongside LLM-based, rule-based, and deep learning (DL)-based methodologies. However, in the repair phase, the comparison focuses on rule-based and LLM-based approaches, as there are currently no suitable DL-based tools available for smart contract repair.

Table I illustrate that our approach has outperformed the rule-based, DL-based, and LLMs-based approaches in the detection phase. (1) First, we used the LLMs-based tool ChatGPT for experiments. We found that the detection performance for reentrancy vulnerabilities was poor, with accuracy, recall, precision, and F1 score values of only 50.96%, 17.31%, 52.94%, and 26.09%, respectively. Additionally, the experiment results of Codellama and ChatGLM are consistent with ChatGPT. This result indicates that directly using LLMs to detect smart contract vulnerabilities performs poorly. (2) Next, we compared our approach with the rule-based tools, and the results showed that our method outperformed the best-performing tool (SliSE) in F1 score. (3) Finally, we compared our approach with DL-based approaches in detection. Among them, AME achieved an accuracy of 90.19%, and ReVulDL showed the best performance in recall, precision, and F1 score, with values of 93%, 92%, and 93%, respectively. However, our method surpassed these deep learning methods in all aspects, with improvements of 3.51%, 2.31%, 0.42%, and 0.85% in accuracy, recall, precision, and F1 score, respectively.

TABLE II
THE OVERALL COMPARISON OF SMART CONTRACT REENTRANCY VULNERABILITY REPAIR.

Method		Fixed	Fixed Rate(%)	Correction	Correction Rate(%)
Rule-Based	SmartFix	46/52	88.46	46/52	88.46
	SGUARD	33/52	63.46	29/52	55.77
	Elysium	35/52	67.31	-	-
LLMs-based	CodeLlama-7B-0shot	10/52	19.23	4/52	7.69
	CodeLlama-7B-2shot	15/52	28.85	6/52	11.54
	CodeLlama-13B-0shot	18/52	34.62	6/52	11.54
	CodeLlama-13B-2shot	25/52	48.07	10/52	19.23
	CodeLlama-13B-instruct-0shot	33/52	63.46	20/52	38.46
	CodeLlama-13B-instruct-2shot	38/52	73.07	24/52	46.15
	ChatGLM3-6B-0shot	30/52	57.69	8/52	15.38
	ChatGLM3-6B-2shot	35/52	67.31	14/52	26.92
	ChatGPT-0shot	36/52	69.23	30/52	57.69
	ChatGPT-2shot	44/52	84.61	42/52	80.77
Our Approach		49/52	94.23(9.62↑)	47/52	90.38(9.61↑)

Table II demonstrates that our approach surpasses both LLMs-based and rule-based approaches in fixing reentrancy vulnerabilities. Initially, we employed various tools to remediate these vulnerabilities and subsequently verified the accuracy of the repaired smart contracts. Among the conventional tools, Smartfix was the most effective, correcting 46 out of 52 vulnerabilities, all of which were successfully repaired. Utilizing ChatGPT without prior repair examples resulted in 36 fixed vulnerabilities, with 30 being successfully repaired. The introduction of fixed example prompt enhanced the effectiveness of LLMs, increasing the number of fixed vulnerabilities to 44, with 42 successfully repaired. Further experiments involved comparisons with CodeLlama-7B, CodeLlama-13B, and ChatGLM3-6B as detailed in Table II. Incorporating our knowledge-aware, prompt-based approach with LLMs markedly improved the repair outcomes, fixing 49 vulnerabilities, with 47 successfully repaired.

2) Answer to RQ2 (The effect of enhanced DL-based detection): In this section, we discuss the impact of our algorithm of feature extraction, data balancing, and pre-trained models on reentrancy vulnerability detection. In feature extraction, we compare our approach with three baselines: direct source code-based input, single-function-based input, and multi-function-based input. In data balancing, we compare imbalanced datasets, datasets balanced with SMOTE, and datasets balanced with our approach. Additionally, we utilize five approaches in cross-language pre-trained models: Word2Vec, Bert, Roberta, CodeBert, and CodeBerta.

Figure 7 illustrates that various feature extraction and data balancing techniques significantly affect the reentrancy vulnerability detection. Our proposed multi-function-based feature extraction and data balancing approaches can improve detection performance. For feature input, the accuracy achieved by source code slices, single-function code slices, and multi-function code slices is 83.28%, 88.93%, and 94.46%, respectively. Additionally, recall, precision, and F1 scores also indicate that training with multi-function code gadgets is significantly better than source code input and single-function code slices. For data balancing, the imbalanced dataset achieves an F1 score of 57.14% and an accuracy of 96.97%, indicating significant overfitting during model training. Conversely, the results trained with SMOTE and our algorithm exceed 90% in both accuracy and F1 scores,

TABLE III
THE OVERALL DETECTION COMPARISON OF VARIOUS PRE-TRAINED MODELS.

Method		Accuracy(%)	Recall(%)	Precision(%)	F1 score(%)
PTM	Word2vec	92.48	93.33	91.8	92.56
	Bert	93.76	94.54	91.6	94.06
	Roberta	93.71	95.97	90.65	94.17
	CodeBert	94.46	95.74	92.17	94.78
	CodeBerta	94.28	95.24	91.5	94.56

demonstrating effective deep learning training. Therefore, we conclude that our data balancing method not only enhances the detection performance of reentrancy vulnerabilities but also mitigates the risk of overfitting in model training.

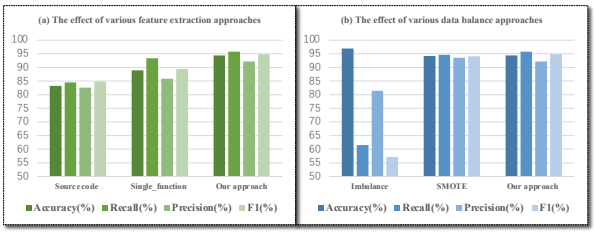

Fig. 7. The effect of various feature extraction and data balance approaches in reentrancy vulnerability detection.

Table III illustrates that different cross-language pre-trained models also impact reentrancy vulnerability detection performance. Pre-trained large language models effectively improve vulnerability detection compared to deep learning models. For instance, the detection accuracy and F1 score achieved by Word2Vec are only 92.48% and 92.56%, while the cross-language pre-trained model CodeBert achieves an accuracy of 94.46% and an F1 score of 94.78%. Comparing the experimental results of Bert and CodeBert, we also find that incorporating code training in pre-trained models can improve reentrancy detection performance to some extent.

3) Answer to RQ3 (The effect of prompt-based vulnerability repair): In this section, we briefly summarize the effectiveness of reentrancy vulnerability repair approaches utilizing LLMs-based tools. We compare the ESC, SmartBugs, and ReentrancyStudy datasets. We conduct comparative experiments with zero-shot prompting for direct repair (ChatGPT-0-shot), two-shot prompting with vulnerability examples and repaired examples (ChatGPT-2-shot), and our hybrid model-driven approach.

Figure 8 shows a vulnerable code fragment repaired by ChatGPT utilizing three approaches: locking-based, replacing-based, and checks-effects-interactions repair. The vulnerability in line 4 can cause a DAO attack (see source code in figure 8). When we seek help from ChatGPT, it can replace the *call.value* function with the transfer function in (b) (see replacing in figure 8) or use lock-based repair in (c) (see locking in figure 8). Finally, in (d) (see CEI in figure 8), the model uses the checks-effects-interactions method to repair the vulnerability: first modifying the contract state in line 4, then interacting with the external contract in line 5, and finally

TABLE IV
THE REPAIR COMPARISON ON ESC, SMARTBUGS AND REENTRANCYSTUDY DATASETS.

Dataset		fixed	Fixed rate(%)	Correct	Correct rate(%)
ESC	ChatGPT-0shot	30/52	57.69	23/52	44.23
	ChatGPT-2shot	44/52	84.61	42/52	80.77
	Our approach	49/52	94.23(9.62↑)	47/52	90.38(9.61↑)
SmartBugs	ChatGPT-0shot	72/174	41.38	63/174	36.21
	ChatGPT-2shot	119/174	68.39	106/174	60.02
	Our approach	156/174	89.66(21.27↑)	140/174	80.46(20.44↑)
ReentrancyStudy	ChatGPT-0shot	32/41	78.05	25/41	60.96
	ChatGPT-2shot	35/41	85.37	31/41	75.61
	Our approach	39/41	95.12(9.75↑)	37/41	90.24(14.63↑)

verifying the success of the interaction in line 6. Thus, we conclude that designing specific prompts for adjustment is beneficial for repairing vulnerabilities, eliminating the need for complex algorithms to repair smart contract vulnerabilities.

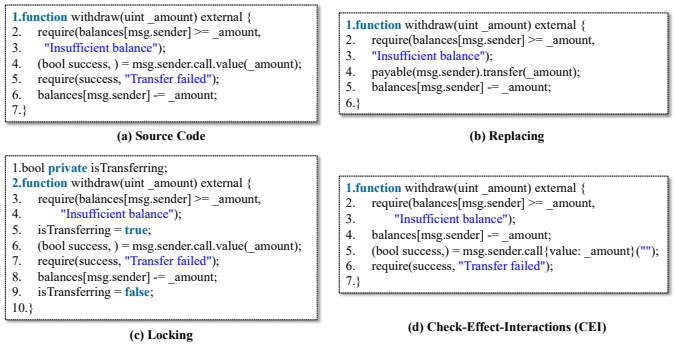

Fig. 8. The examples of source code and three fixed examples.

As demonstrated in Table IV, enhancing the input prompts considerably improves the performance of LLMs in repairing vulnerabilities across the ESC, SmartBugs, and ReentrancyStudy datasets. Initially, in the zero-shot setting (0-shot), where ChatGPT solely evaluates and analyzes vulnerable smart contracts from the ESC dataset, the results are suboptimal: ChatGPT successfully repairs 30 code slices, with only 23 being correct. However, significant improvements are observed when ChatGPT is provided with examples of repairs and corresponding methods in the prompt (2-shot), leading to the successful repair of 44 code slices, with 42 being accurately repaired. The most pronounced enhancement is noted when ChatGPT is further equipped with a knowledge-aware prompt, culminating in the successful repair of 48 code slices, with 47 being correct. Notably, the results on the SmartBugs and ReentrancyStudy datasets align with those observed in the ESC dataset. These findings underscore the efficacy of incorporating vulnerability examples into the model's inputs, as well as the utility of attention scores-based code gadgets in enhancing the model's ability to precisely locate and amend vulnerabilities.

VI. DISCUSSION

A. The effect of prompt type for repair

In Figure 9, the influence of different types of repair examples included in the prompts on experimental performance is evident. Specifically, the nature of the examples

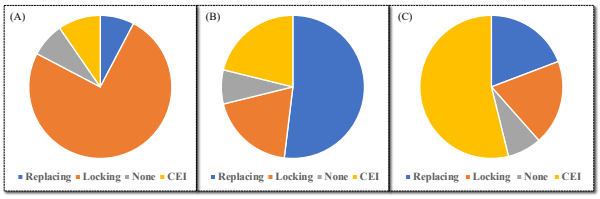

Fig. 9. The distribution analysis of different prompt types in vulnerability repair.

TABLE V
THE IMPACT OF THE ORDER OF REPAIR EXAMPLES ON THE REPAIR EXPERIMENT

Order of prompt	Unchecked	Locking	Replacing	CEI
Vulnerable Example + Locking + Replacing + CEI	3	28	10	11
Vulnerable Example + Replacing + Locking + CEI	3	26	19	4
Vulnerable Example + CEI + Locking + Replacing	3	29	10	10

has a substantial impact on the model's proficiency in addressing vulnerabilities. For instance, prompts that incorporate both vulnerability and lock-based repair examples lead to a predominance of lock-based solutions, with 39 out of 52 vulnerability cases resolved using this method, as depicted in Figure 9. Similarly, when the prompts include vulnerability examples alongside function replacement repair examples, the model tends to generate more function replacement solutions, successfully repairing 27 out of 52 samples. Additionally, prompts that combine vulnerability examples with checks-effects-interactions (CEI) repair examples result in 28 out of 52 samples being remedied through the CEI strategy.

B. The effect of prompt order for repair

Table V illustrates that the sequence in which input examples are provided influences the model's outcomes, with lock-based repairs persisting as the predominant repair strategy. Specifically, the frequency of lock-based repairs surpasses that of function replacement and condition-exit-immediate (CEI) repairs across all evaluated orders of repair prompts. This observation suggests that ChatGPT demonstrates a preference for employing lock-based methods to repair vulnerabilities.

VII. Threats to Validity

Vulnerability Types: Our approach is exclusively applied to reentrancy vulnerability. As the Ethereum ecosystem continues to reveal more vulnerabilities, it is imperative to conduct further research on diverse vulnerabilities, including timestamp dependency, infinite loop, access control, and unchecked external calls vulnerability. Additionally, reentrancy vulnerabilities can stem from intricate sources, and existing works have limited capability [48], researchers should uncover and detect more reentrancy patterns beyond those solely associated with *call.value*.

LLMs-driven smart contract safety: We have recently implemented a limited knowledge-aware prompt tuning to enhance the performance of LLMs. However, given their limited feature coverage of reentrancy vulnerabilities, there must be more untapped potential that can be found by utilizing the large language model. Additionally, various rules and algorithms are available to extract multi-function code gadgets and augment the imbalanced datasets, which restrict the application of LLMs. Therefore, incorporating LLMs into the solution may offer a more convenient approach to solving this problem.

VIII. Conclusion

In this paper, we introduce a unified hybrid framework that integrates deep learning (DL)-based and knowledge-aware LLMs-based methodologies to improve the detection and repair of reentrancy vulnerabilities in smart contracts. This framework mitigates the inherent limitations of individual models by enhancing the feature extraction and balancing of dataset issues in smart contract code. Specifically, we develop a novel feature extraction algorithm to improve the identification of vulnerability indicators and implement a data balancing algorithm that preserves both the syntactic and semantic integrity of code slices. Additionally, we construct a neural network architecture that leverages cross-language pre-trained models to refine code representation and vulnerability classification. For the repair phase, we apply deep learning-driven attention mechanisms and domain-specific knowledge to generate targeted repair prompts. Our comprehensive experimental evaluation demonstrates that our framework surpasses existing benchmarks, thereby significantly advancing the security framework for blockchain systems.

In future work, we will extend our model's capabilities to other LLM-driven smart contract vulnerabilities and explore advanced optimization techniques for enhancing the reasoning abilities of large language models in this domain.

Acknowledgements

This research was supported by the National Science Foundation of China (No.62302437).

References

[1] Y. Pan, Z. Xu, L. T. Li, Y. Yang, and M. Zhang, "Automated generation of security-centric descriptions for smart contract bytecode," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 1244–1256.

[2] T. Chen, R. Cao, T. Li, X. Luo, G. Gu, Y. Zhang, Z. Liao, H. Zhu, G. Chen, Z. He *et al.*, "Soda: A generic online detection framework for smart contracts." in *NDSS*, 2020.

[3] Z. Zheng, N. Zhang, J. Su, Z. Zhong, M. Ye, and J. Chen, "Turn the rudder: A beacon of reentrancy detection for smart contracts on ethereum," 2023. [Online]. Available: https://arxiv.org/abs/2303.13770

[4] Z. Zhang, Y. Lei, M. Yan, Y. Yu, J. Chen, S. Wang, and X. Mao, "Reentrancy vulnerability detection and localization: A deep learning based two-phase approach," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–13.

[5] P. Zhang, F. Xiao, and X. Luo, "Soliditycheck: Quickly detecting smart contract problems through regular expressions," *arXiv preprint arXiv:1911.09425*, 2019.

[6] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: analyzing safety of smart contracts." in *Ndss*, 2018, pp. 1–12.

[7] 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, May 2019. [Online]. Available: http://dx.doi.org/10.1109/WETSEB.2019.00008

[8] S. Shakya, A. Mukherjee, R. Halder, A. Maiti, and A. Chaturvedi, "Smartmixmodel: machine learning-based vulnerability detection of solidity smart contracts," in *2022 IEEE international conference on blockchain (Blockchain)*. IEEE, 2022, pp. 37–44.

[9] C. Chen, J. Su, J. Chen, Y. Wang, T. Bi, Y. Wang, X. Lin, T. Chen, and Z. Zheng, "When chatgpt meets smart contract vulnerability detection: How far are we?" *arXiv preprint arXiv:2309.05520*, 2023.

[10] 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.

[11] M. Fu, C. K. Tantithamthavorn, V. Nguyen, and T. Le, "Chatgpt for vulnerability detection, classification, and repair: How far are we?" in *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2023, pp. 632–636.

[12] I. David, L. Zhou, K. Qin, D. Song, L. Cavallaro, and A. Gervais, "Do you still need a manual smart contract audit?" *arXiv preprint arXiv:2306.12338*, 2023.

[13] 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*, 2016, pp. 254–269.

[14] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 67–82.

[15] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," in *Proceedings of the 1st international workshop on emerging trends in software engineering for blockchain*, 2018, pp. 9–16.

[16] Z. Wang, J. Chen, Y. Wang, Y. Zhang, W. Zhang, and Z. Zheng, "Efficiently detecting reentrancy vulnerabilities in complex smart contracts," *arXiv preprint arXiv:2403.11254*, 2024.

[17] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "Smartbugs: A framework to analyze solidity smart contracts," in *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, 2020, pp. 1349–1352.

[18] 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. 19 685–19 695, 2020.

[19] M. Rodler, W. Li, G. O. Karame, and L. Davi, "{EVMPatch}: Timely and automated patching of ethereum smart contracts," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1289–1306.

[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*, 2021, pp. 3283–3290.

[21] 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*, 2021.

[22] Z. Liu, P. Qian, X. Wang, L. Zhu, Q. He, and S. Ji, "Smart contract vulnerability detection: from pure neural network to interpretable graph feature and expert pattern fusion," *arXiv preprint arXiv:2106.09282*, 2021.

[23] Z. Li, M. Pan, Y. Pei, T. Zhang, L. Wang, and X. Li, "Robust learning of deep predictive models from noisy and imbalanced software engineering datasets," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–13.

[24] T. Chakraborty and A. K. Chakraborty, "Hellinger net: A hybrid imbalance learning model to improve software defect prediction," *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 481–494, 2020.

[25] P. R. Bal and S. Kumar, "Wr-elm: Weighted regularization extreme learning machine for imbalance learning in software fault prediction," *IEEE Transactions on Reliability*, vol. 69, no. 4, pp. 1355–1375, 2020.

[26] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE transactions on software engineering*, vol. 47, no. 10, pp. 2084–2106, 2019.

[27] N. Ivanov, C. Li, Q. Yan, Z. Sun, Z. Cao, and X. Luo, "Security threat mitigation for smart contracts: A comprehensive survey," *ACM Computing Surveys*, vol. 55, no. 14s, pp. 1–37, 2023.

[28] P. Qian, Z. Liu, Q. He, B. Huang, D. Tian, and X. Wang, "Smart contract vulnerability detection technique: A survey," *arXiv preprint arXiv:2209.05872*, 2022.

[29] 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, 2020.

[30] M. Li, X. Ren, H. Fu, Z. Li, and J. Sun, "Convmhsa-scvd: Enhancing smart contract vulnerability detection through a knowledge-driven and data-driven framework," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 578–589.

[31] Y. Zhang, S. Ma, J. Li, K. Li, S. Nepal, and D. Gu, "Smartshield: Automatic smart contract protection made easy," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 23–34.

[32] 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)*. IEEE, 2021, pp. 1–8.

[33] P. Qian, Z. Liu, Y. Yin, and Q. He, "Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2220–2229.

[34] X. L. Yu, O. Al-Bataineh, D. Lo, and A. Roychoudhury, "Smart contract repair," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 4, pp. 1–32, 2020.

[35] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[36] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[37] L. Hillebrand, A. Berger, T. Deußer, T. Dilmaghani, M. Khaled, B. Kliem, R. Loitz, M. Pielka, D. Leonhard, C. Bauckhage *et al.*, "Improving zero-shot text matching for financial auditing with large language models," in *Proceedings of the ACM Symposium on Document Engineering 2023*, 2023, pp. 1–4.

[38] W. M. I.-C. L. Work, "Rethinking the role of demonstrations: What makes in-context learning work?"

[39] X. Ren, X. Ye, D. Zhao, Z. Xing, and X. Yang, "From misuse to mastery: Enhancing code generation with knowledge-driven ai chaining," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 976–987.

[40] Y. Xue, M. Ma, Y. Lin, Y. Sui, J. Ye, and T. Peng, "Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 1029–1040.

[41] C. F. Torres, J. Schütte, and R. State, "Osiris: Hunting for integer bugs in ethereum smart contracts," in *Proceedings of the 34th annual computer security applications conference*, 2018, pp. 664–676.

[42] S. So and H. Oh, "Smartfix: Fixing vulnerable smart contracts by accelerating generate-and-verify repair using statistical models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 185–197.

[43] T. D. Nguyen, L. H. Pham, and J. Sun, "Sguard: towards fixing vulnerable smart contracts automatically," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1215–1229.

[44] C. Ferreira Torres, H. Jonker, and R. State, "Elysium: Context-aware bytecode-level patching to automatically heal vulnerable smart contracts," in *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*, 2022, pp. 115–128.

[45] Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, and J. Tang, "Glm: General language model pretraining with autoregressive blank infilling," *arXiv preprint arXiv:2103.10360*, 2021.

[46] 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.

[47] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[48] Z. Zheng, N. Zhang, J. Su, Z. Zhong, M. Ye, and J. Chen, "Turn the rudder: A beacon of reentrancy detection for smart contracts on ethereum," 2023.