# SolGPT: a GPT-Based Static Vulnerability Detection Model for Enhancing Smart Contract Security *

Shengqiang Zeng[1,2], Hongwei Zhang[1,2](✉), Jinsong Wang[1,2], and Kai Shi[1,2]

[1] School of Computer Science and Engineering, Tianjin University of Technology, Tianjin 300384, China
poilzero@stud.tjut.edu.cn, hwzhang@email.tjut.edu.cn, {jswang,shikai0229}@tjut.edu.cn

[2] National Engineering Laboratory for Computer Virus Prevention and Control Technology, Tianjin University of Technology, Tianjin 300457, China

**Abstract.** In this study, we present SolGPT, a novel approach to addressing the pivotal issue of detecting and mitigating vulnerabilities inherent in smart contracts, particularly those written in Solidity, the predominant language for smart contracts. Conventional deep learning methodologies largely rely on an abundant pool of labeled training instances, a resource that remains scarce in the domain, thereby limiting the efficacy of vulnerability detection. SolGPT seeks to counteract this limitation by employing an augmented GPT-2 architecture uniquely tailored for smart contract analysis. The model is enriched by Solidity Adaptive Pre-Training to amplify its feature extraction prowess, hence, reducing the reliance on copious amounts of labeled samples. SolGPT further enhances its field-specific adaptation via the introduction of SolTokenizer, a specialized tokenizer devised for smart contracts, thereby augmenting tokenization precision and efficiency. Subsequently, the model is refined to proficiently pinpoint known vulnerabilities in smart contracts, thereby offering real-time vulnerability detection and prescribing preventive countermeasures. Comprehensive evaluation demonstrates that SolGPT outperforms the state-of-the-art detection techniques in terms of accuracy, F1 score, and two other pertinent performance metrics. Notably, when compared to the best-performing alternative among the four vulnerabilities, SolGPT exhibits an average accuracy improvement of 12.85% and an average F1 score improvement of 18.55%. Consequently, the results underscore the potential of SolGPT in substantially advancing the security framework of the blockchain ecosystem.

**Keywords:** Smart Contract · GPT · Vulnerability detection · Adaptive Pre-Training · Blockchain.

* Supported by the National Natural Science Foundation of China (Grant No. 62072336), the Tianjin New Generation of Artificial Intelligence Science and Technology Major Project (Grant No. 19ZXZNGX00080).

## 1 Introduction

With blockchain technology continuing to gain widespread adoption[1], it has exposed numerous security challenges. Smart contracts, a key component of blockchain applications[2], play a critical role in various industries. However, the pervasive presence of vulnerabilities in smart contracts, such as reentrancy, timestamp, delegate call, and integer overflow attacks, has hindered the pace of development and raised significant concerns regarding their security[3]. Consequently, the need for devising efficient techniques to detect and prevent these vulnerabilities in smart contracts has become a vital research direction.

Traditional smart contract vulnerability detection approaches, such as rule-based approaches[4], formal verification[5, 6], symbolic execution[7, 8], and fuzz testing[9, 10], have been employed with varying degrees of success. However, they suffer from limitations including reliance on expert-defined rules, steep learning curves, path explosion, and the inability to exhaust all inputs. These drawbacks motivate the exploration of deep learning approaches to enhance the detection of vulnerabilities in smart contracts.

Deep learning has demonstrated remarkable potential across various security fields[11], encompassing malware detection and classification[12], vulnerability detection[13], intrusion detection[14], and privacy security[15]. In the smart contract vulnerability detection field, existing deep learning-based approaches are limited by the availability of labeled data, and they do not fully exploit the benefits of unsupervised pre-training and transfer learning[16–20]. The GPT series models already attained remarkable triumph in natural language processing (NLP) assignments due to their unsupervised pre-training and transfer learning capabilities[21–23]. These advantages make GPT models an attractive solution to address the challenges in smart contract vulnerability detection.

Inspired by these successes, SolGPT proposes a solidity adaptive pre-training approach for the smart contract security field using the GPT-based feature extracting structure. SolGPT also develop a specialized tokenizer, SolTokenizer, for smart contract-specific keywords. This evaluation across four smart contract vulnerability detections demonstrates SolGPT's superior performance compared to advanced benchmarks. SolGPT highlights the potential of unsupervised pre-training and transfer learning for improving smart contract vulnerability detection.

The major contributions of SolGPT can be delineated as follows:

1. Adapting the GPT architecture to the field of smart contract vulnerability detection, significantly enhancing performance metrics through the use of transfer learning, and addressing the challenge of limited labeled data.
2. Employing Solidity Adaptive Pre-Training and Vulnerability Detection Fine-tuning for the smart contract vulnerability detection, enabling the GPT-based model to better capture Solidity-specific features and further improve transfer learning performance.
3. Developing a specialized SolTokenizer, specifically designed for the smart contract field, resulting in improved tokenization performance.

## 2 Problem Formulation

In the context of blockchain technology and smart contract security, the fundamental problem addressed in this research can be formally defined as follows:

**Given**: A labeled dataset $\mathcal{L}$ containing pairs $(C_i, y_i)$, where $C_i$ represents a source smart contract code and $y_i$ is a binary label where $y_i = 1$ represents $C_i$ has a vulnerability of a certain type while $y_i = 0$ denotes $C_i$ is safe.

**Formulation**: Given a smart contract $C_i$, SolGPT will produce a binary classification output $\hat{y}_i$ that predicts whether $C_i$ is vulnerable or not. Mathematically, this can be expressed as:

$$\hat{y}_i = \text{SolGPT}(C_i) \tag{1}$$

Where: $\hat{y}_i \in \{0, 1\}$ represents the predicted vulnerability status of the smart contract $C_i$. SolGPT$(C_i)$ denotes the prediction made by the SolGPT model for the input smart contract $C_i$.

**Performance Metrics**: The evaluation of SolGPT will be based on a set of performance metrics. These metrics, detailed in the experimental section, include but are not limited to accuracy, precision, recall, and F1-score.

The problem of smart contract vulnerability detection is inherently challenging due to the diversity of vulnerabilities and the limited availability of labeled data. SolGPT addresses these challenges by adapting the GPT architecture, employing Solidity Adaptive Pre-Training, and utilizing specialized tokenization techniques, ultimately aiming to achieve superior performance in identifying vulnerabilities in smart contracts.

## 3 Related Work

In this section, current deep learning-based approaches for smart contract vulnerability detection are discussed, and their limitations are analyzed. The motivation behind the proposed approach is introduced, drawing inspiration from the GPT models.

### 3.1 Deep Learning-Based approaches for Smart Contract Vulnerability Detection

Deep learning-based approaches for smart contract vulnerability detection have been developed as an alternative to traditional approaches, aiming to improve performance by learning from labeled samples. Some of the notable works in this area include:

**SaferSC[16]** as an early application of deep learning to smart contract vulnerability detection, SaferSC utilized an LSTM (Long Short-Term Memory) network to enhance the Traditional tool Maian for more effective detection. This approach resulted in improved accuracy compared to traditional approaches, but it does not get rid of the drawbacks of traditional approaches.

**ReChecker[17] and ContractWard[18]** proposed a scheme for detecting vulnerabilities in smart contracts based on Word2Vec embeddings[24], improved recurrent neural networks, and attention mechanisms[25]. These schemes completely break away from the traditional approach for the first time, completely use the approach of deep learning, and achieve good results in some vulnerabilities.
**DR-GCN[19]** introduced the Graph Convolutional Neural Network (GCN) to detection and its improved information propagation, Temporal Message Passing (TMP). These approaches employed a contract graph representation and demonstrated excellent detection performance for certain vulnerabilities.
**GCN with Expert Knowledge[20]** proposes an approach for detecting smart contract vulnerabilities using graph neural networks and expert knowledge, which showed significant accuracy improvements on three types of vulnerabilities in Ethereum and VNT Chain platforms.

Notwithstanding the benefits of deep learning techniques in certain aspects, there remains significant room for improvement. One issue is that the neural network architectures employed in existing deep learning-based approaches may not effectively capture the contextual information in the code. This can lead to the potential loss of semantic information, contributing to the suboptimal performance of these approaches in vulnerability detection tasks.

Another key limitation of these deep learning approaches is their substantial reliance on the number of labeled samples. In the domain of smart contract vulnerability detection, the number of labeled samples is relatively limited compared to other domains, hindering the effectiveness of these approaches in improving performance.

Deep learning-based approaches also face challenges during the tokenization process when dealing with smart contract code. During tokenization, semantic blocks may be unintentionally split into different parts, causing a loss of complete semantic information. This issue could further impact vulnerability detection performance by reducing the model's understanding of the code structure and semantics.

### 3.2 GPT Models

To overcome these limitations, attention can be turned to the GPT (Generative Pre-trained Transformer) series of models[21–23], originally developed by OpenAI. GPT models have demonstrated significant success in various NPL assignments, such as sentiment analysis, text summarization, and text generation. The GPT architecture is built upon the Transformer model[26], which was introduced by Vaswani et al. in 2017 and has since become a cornerstone in the NLP field due to its outperforming scalability and high-level attention mechanism.

The GPT models employ unsupervised pre-training, where the model learns the statistical properties of the input data in unsupervised learning. And followed by transfer learning, where the pre-trained model is fine-tuned on a specific task using labeled data. This two-step process enables GPT models to utilize the

knowledge learned from pre-training and adapt it to downstream assignments with relatively fewer labeled samples.

## 4 SolGPT: Solidity Adapative GPT

In this section, a novel deep learning-based approach called SolGPT is presented to improve vulnerability detection in Solidity smart contracts. SolGPT addresses the challenges faced by existing deep learning-based approaches by introducing the GPT-2 architecture, employing a solidity adaptive pre-training approach, and designing a specialized tokenizer. By these approaches, SolGPT aims to mitigate the dependency on a large quantity of labeled samples and further enhance the performance of smart contract vulnerability detection.

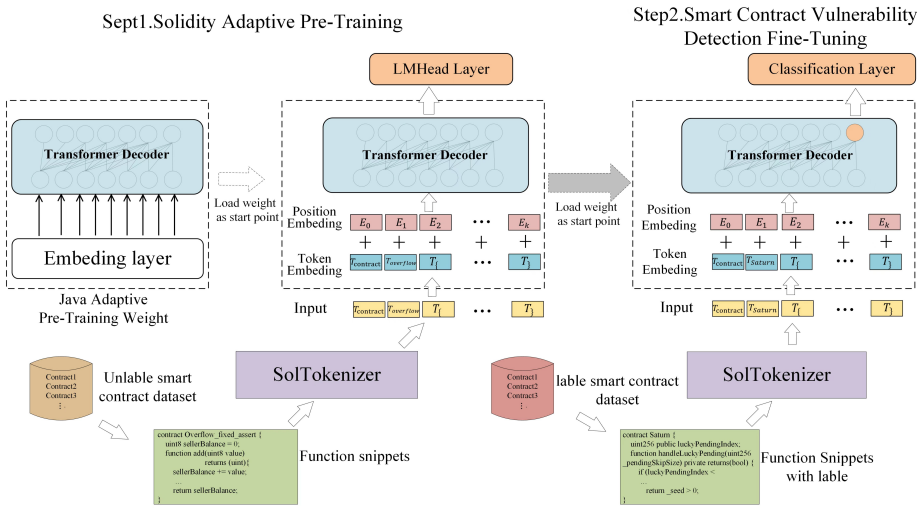


**Fig. 1.** Training Procedure.

The construction of SolGPT involves two training steps, Solidity Adapative Pre-Training, and Smart Contract Vulnerability Detection Fine-Tuning, with transfer learning applied between these two steps to transfer the model weights across different training tasks, as indicated by the dashed arrows in the Figure 1. The feature extraction component of the model (enclosed by the dashed box in the Figure 1) remains consistent across both two steps, leveraging the GPT-2 Base architecture (Parameters: 117M, Transformer Decoder Layers: 12, Hidden Size: 768) to extract relevant features from the smart contract code. The LM-Head and ClassificationHead are task-specific heads, consisting of simple fully connected neural layers and training algorithms tailored to each training task. Furthermore, SolGPT introduce a specialized SolTokenizer to enhance the tokenization performance for smart contract segments. The details of this compo-

nent are elaborated in Section 3.1. These two steps and the corresponding model structure improve the model's generalization capabilities and task performance, and the details of these two steps are discussed in Sections 3.2 and 3.3.

In summary, Figure 1 provides a comprehensive visualization of the entire SolGPT training process, showcasing the adaptation of GPT-2's feature extraction capabilities to the Solidity programming language.

### 4.1 SolTokenizer

In order to bolster SolGPT's ability to comprehend Solidity code, we employ a novel and specialized tokenizer, the SolTokenizer. This refinement ensures enhanced tokenization performance for Solidity, a distinct language with unique syntactic and structural properties that poses distinctive challenges for standard tokenization algorithms.

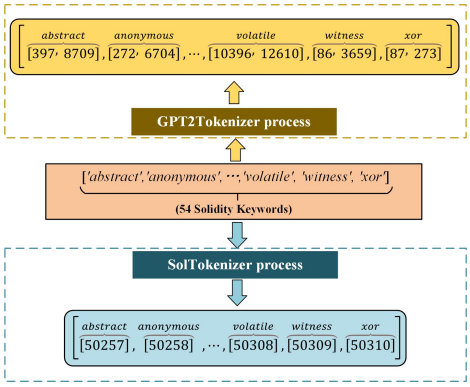


**Fig. 2.** Difference between GPT2Tokenizer and SolTokenizer.

Drawing inspiration from the Byte-Pair Encoding (BPE)[27] tokenization algorithm, a groundbreaking advancement in the field of Neural Machine Translation, SolTokenizer combines the technical expertise of BPE with a strategic approach to keyword tokenization. This amalgamation empowers SolTokenizer to deliver precise and refined tokenization results for smart contract codes, a challenging task due to their intricate nature and distinct characteristics.

Fundamentally, SolTokenizer leverages the BPE algorithm, enabling it to segment contract code not only at the character level but also into smaller sub-word sequences. This meticulous dissection uncovers subtle nuances and semantic information embedded within contract codes, thereby enhancing the performance of subsequent analytical tasks. In comparison to traditional character-level tokenization, SolTokenizer offers a more precise and comprehensive representation of the code.

Moreover, SolTokenizer introduces a novel approach by imposing tokenization restrictions on 54 crucial keywords commonly found within smart contracts.

These keywords are treated as indivisible units, ensuring they remain intact without further subdivision into sub-words. This restriction strengthens the integrity and accuracy of these keywords, enabling the GPT network to intuitively comprehend and identify essential terminology within the contract codes, eliminating the need for additional semantic inference.

To summarize, SolTokenizer serves as an advanced tokenizer for smart contracts, merging the BPE algorithm with a strategic keyword tokenization approach to yield accurate, detailed, and semantically consistent tokenization results for GPT. This state-of-the-art tokenization technique not only enhances the understanding and analysis of smart contract code but also establishes a robust foundation for future research endeavors.

### 4.2 Solidity Adaptive Pre-training

To address the challenge posed by the limited number of labeled samples in smart contract vulnerability detection, the Solidity Adaptive Pre-training approach for SolGPT will be introduced, which incorporates an unsupervised pre-training step through modeling the high-level semantics of smart contracts. Specifically, SolGPT begins with CodeGPT-small-java, a GPT-2 pre-trained weights on a large Java code corpus by Microsoft, as a strong foundation. However, to account for the differences in syntax and semantics between Solidity and Java, SolGPT performs additional pre-training on Solidity code for better downstream smart contract vulnerability detection performance.

In this training step, the approach uses the same dataset as in the fine-tuning phase (detailed information will be provided in the specific experimental section), but without utilizing the labeled information indicating the presence of vulnerabilities in the dataset. Utilizing additional unlabeled datasets for training during this phase can further enhance generalization, resulting in improved performance on fine-tuning tasks. Moreover, in actuality, the vast majority of accessible data lacks labels, with only a limited fraction being labeled. Hence, unlabeled samples are plentiful and readily obtainable.

Prior to commencing the training process, weights from CodeGPT-java[28] are loaded as the initial state for the model's learning. This serves as the starting point for the subsequent training steps. Thereafter, the intelligent contract segment is introduced into the model for the extraction of features, resulting in token-level feature vectors. Utilizing the LMHead layer, the nth feature vector is translated into its associated token within the vocabulary, signifying the prediction of the nth+1 token derived from this vector. In the concluding stage, the model computes the cross-entropy loss between the predicted token and the true nth+1 token present in the segment (with n ranging from 1 to x-1 for every code segment incorporating x tokens), followed by backpropagation for the training process. The Solidity Adaptive Pre-training mechanism is delineated as Algorithm 1.

In Solidity Adaptive Pre-Training, SolGPT uses gradient descent optimization to update the model's parameters during the backpropagation. Specifically, SolGPT updates the vulnerability-specific parameter $\theta_{SolidityVulnerability}$ using

**Algorithm 1** Solidity Adaptive Pre-training

```
Initialization: Load SolGPT with CodeGPT-small-java weights
Initialization: Load MPL ← Linear(hidden layer size, vocabulary size)
for each epoch in pre-training epochs do
    for each batch in dataset do
        tokens ← SolTokenizer(batch)
        labels ← tokens
        feature_vectors ← SolGPT(tokens)
      LMHead layer for predicting tokens:
        lm_logits ← MPL(feature_vectors)
        lm_logits ← Softmax(lm_logits)
      Calculate language modeling loss by comparing predicted tokens with real
      tokens:
        shift_logits ← lm_logits[:-1]
        shift_labels ← labels[1:]
        loss ← CrossEntropyLoss(shift_logits, shift_labels)
      Update model weights with gradient descent
    end for
end for
Save pre-trained SolGPT model
```

**Input:** Solidity dataset for pre-training
**Output:** Pre-trained SolGPT model

the gradient of the cross-entropy loss function with respect to the model's parameters $\theta$. The general domain parameter $\theta_{GeneralJavaDomain}$ is also used during the update process. The optimization formula is as follows:

$$\theta_{SolidityVulnerability} = \theta_{GeneralJavaDomain} - \alpha \nabla_{\theta} Lce \tag{2}$$

Here, $\alpha$ is the learning rate, which determines the step size of each parameter update. The gradient of the cross-entropy loss function $\nabla_{\theta} Lce$ indicates the direction and magnitude of the change in the loss function with respect to each parameter. It measures how much each parameter affects the output of the model and is used to adjust the values of the parameters in order to minimize the loss function.

After completing the Solidity adaptive pre-training, SolGPT proceed with the fine-tuning process for smart contract vulnerability detection, as described in the "Vulnerability Detection Fine-tuning" subsection. The Solidity adaptive pre-training helps the model to better generalize to the vulnerability detection task, ultimately leading to improved performance in detecting vulnerabilities in Solidity smart contracts.

### 4.3 Vulnerability Detection Fine-tunning

To assess the efficacy of SolGPT in detecting vulnerabilities in smart contracts, SolGPT chose four common types of smart contract vulnerabilities. To achieve this, SolGPT established a basic classification layer atop the pre-existing SolGPT

model. This classification layer was composed of a fully connected layer and a softmax layer. By fine-tuning the SolGPT model through this process, SolGPT acquired a fine-tuned version of SolGPT that was optimized for this specific objective.

In vulnerability detection fine-tuning, SolGPT uses the hidden states produced by SolGPT, denoted as $SolFeature_i$, as inputs to an additional structure classification head. This layer is designed to map the high-dimensional hidden states to a lower-dimensional space for classification purposes. Specifically, SolGPT defines the logits of the classification head as a softmax function applied to the linear transformation of $SolFeature_i$, represented as:

$$logits = \text{softmax}(W_h SolFeature_i + b_h) \tag{3}$$

Here, $W_h$ and $b_h$ are the weight matrix and bias vector of a multi-layer perceptron (MLP) layer that projects the 768-dimensional hidden layer feature vectors onto 2-dimensional vectors for vulnerability detection. The softmax function normalizes the output logits into a probability distribution over the existence of the specific smart contract vulnerability, allowing us to make a prediction for the input solidity code. The softmax function can be expressed as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{4}$$

where $x_i$ represents the $i$th element of the logits, and $n$ is the length of the logits.

During the fine-tuning phase, SolGPT used a dataset of smart contracts containing both vulnerable and non-vulnerable code. The corresponding code was first inputted into SolGPT with Classification Head to predict whether specific vulnerabilities existed. The predicted results were then compared with the actual labels, and the backpropagation algorithm was used to update the model parameters. The Solidity Fine-Tunning procedure can be described as Algorithm 2.

After completing the vulnerability detection fine-tuned, the fine-tuned SolGPT is evaluated by some experiments as described in section 3 "Experiment And Analysis" which show significantly outperformed the baseline GPT-2 and cutting-edge smart contract vulnerability detection approaches in identifying vulnerabilities in smart contracts. The performance improvement is mainly due to the SolTokenizer, the Solidity Adaptive Pre-Training, and the vulnerability detection fine-tuning that SolGPT applied, which enables the SolGPT to better understand the unique syntax and structure of Solidity.

## 5 Experiment and Analysis

In this section, to evaluate the effectiveness of SolGPT in detecting various smart contract vulnerabilities, the fine-tuned SolGPT is constructed using the last section mentioned approach to detect four specific vulnerabilities and compare its performance with existing approaches. Additionally, ablation studies are conducted to analyze the contributions of different components of this proposed

**Algorithm 2** Vulnerability Detection Fine-tuning

```
Initialization: Load pre-trained SolGPT model
Initialization: Load MPL ← Linear(hidden layer size, 2)
for each epoch in fine-tuning epochs do
    for each batch in dataset do
        tokens ← SolTokenizer(batch)
        labels ← vulnerability labels (0 or 1) for contracts in a batch
        feature_vectors ← SolGPT(tokens)
        lm_logits ← MPL(feature_vectors)
        lm_logits ← Softmax(lm_logits)
      Calculate vulnerability detection loss by comparing predicted labels with real
      labels:
        loss ← CrossEntropyLoss(logits, labels)
      Update model weights with gradient descent
    end for
end for
Save fine-tuned SolGPT model
```

**Input:** vulnerability detection dataset (vulnerable and non-vulnerable contracts)
**Output:** Fine-tuned SolGPT model for vulnerability detection

approach, including the specialized tokenizer, SolTokenizer, and Solidity Adaptive Pre-Training. Especially, the term "fine-tuned SolGPT" will be replaced by "SolGPT" in the following descriptions to avoid redundancy and improve readability.

**Runtime Environment:** In these experiments, the runtime environment can be described as Table 1.

**Table 1.** Fine-tuning runtime environment.

| Category | Adaptive Pre-Training |
|---|---|
| Deep Learning Framework | Pytorch |
| System | Ubuntu22.04 LTS |
| CPU Processor | Intel Core i5-12400 CPU |
| GPU Processor | NVIDIA Tesla P40 24G |
| Memory Capacity | 16 GB RAM |

**Experimental Hyperparameters:** This study utilizes cross-validation to evaluate the impact of various experimental hyperparameters on the outcomes. Through the comparison of diverse hyperparameter settings, the optimal parameters are eventually determined. The ultimate experimental parameters are outlined in Table 2.

**Datasets:** Experiment utilized the recently released public Smart-Contract-Dataset[29]. In particular, the Resource2 dataset was meticulously chosen as the primary source for analysis due to its availability and suitability for evaluation purposes. While Resource1 comprises an extensive dataset, its unlabeled nature

**Table 2.** Selection of fine-tuning hyperparameters.

| Hyperparameter | Meaning | Value |
|---|---|---|
| Learning Rate | initial learning rate | 5e-5 |
| Batch Size | training sample amount per batch | 4 |
| Epoch | training rounds | 50 |
| Optimizer | learning rate optimization algorithm | AdamW |
| Dropout | random inactivation rate | 0.1 |
| Embed Size | word vector dimension | 768 |

renders it unsuitable for direct assessment. Moreover, Resource3 was published subsequent to the experiment, prompting the deliberate selection of Resource2 for this investigation.

The flawed-marked dataset used in this study is derived from contract code crawled from Ethereum. The author of the dataset extracted code snippets related to vulnerabilities through automated control flow analysis and manually labeled the snippets. All experiments in this paper will be based on this dataset. The approach employed in the dataset to extract vulnerability-related code snippets through automated control flow analysis, as originally proposed in the 2018 paper "Vuldeepecker" [30], will not be further elaborated upon in this context.

In regard to the dataset size, we acknowledge that the selected dataset, particularly the Resource2 dataset, may have limitations in terms of the number of types and samples. The primary focus of our proposed method is to address the challenge of a scarcity of labeled samples, which is a common issue encountered in similar datasets during model training. Our approach incorporates techniques and strategies to effectively handle the limited dataset size, aiming to demonstrate robustness and generalization despite this constraint. We emphasize that the goal of this paper is to showcase the effectiveness and efficiency of our method, even under circumstances where the dataset is not extensive.

Table 3 presents the four subsets of the dataset, each corresponding to a specific vulnerability. It includes the number and percentage of positive samples (samples with the vulnerability) and negative samples (samples without the vulnerability) in each subset, as well as the total number of samples in each subset. The percentages are calculated based on the total number of samples in each subset. Prior to experimentation, the dataset was divided into a training set and a verification set according to a 7:3 ratio. The division ensured the balance of positive and negative samples proportion in both sets, mitigating issues that may arise from sample proportion imbalance caused by direct random division.
**Evaluation Metrics:** SolGPT employ commonly used evaluation metrics in the vulnerability detection field, including accuracy, precision, recall, and F1-Score. The formulas for these metrics are as follows:

- $Accuracy = (TP + TN)/(TP + TN + FP + FN)$.
- $Precision = TP/(TP + FP)$.
- $Recall = TP/(TP + FN)$.
- $F1 - Score = 2 \times (Precision \times Recall)/(Precision + Recall)$.

**Table 3.** Dataset sub-set positive and negative sample distribution for different vulnerability types.

| Vulnerability Type | Positive Samples | Negative Samples | Total Samples |
|---|---|---|---|
| reentrancy | 73 (26.74%) | 200 (73.26%) | 273 |
| timestamp | 179 (51.29%) | 170 (48.71%) | 349 |
| delegatecall | 62 (31.63%) | 134 (68.37%) | 196 |
| integeroverflow | 90 (32.73%) | 185 (67.27%) | 275 |

Here, TP represents true positives, TN represents true negatives, FP represents false positives, and FN represents false negatives.

### 5.1 Comparisons with Traditional Approaches

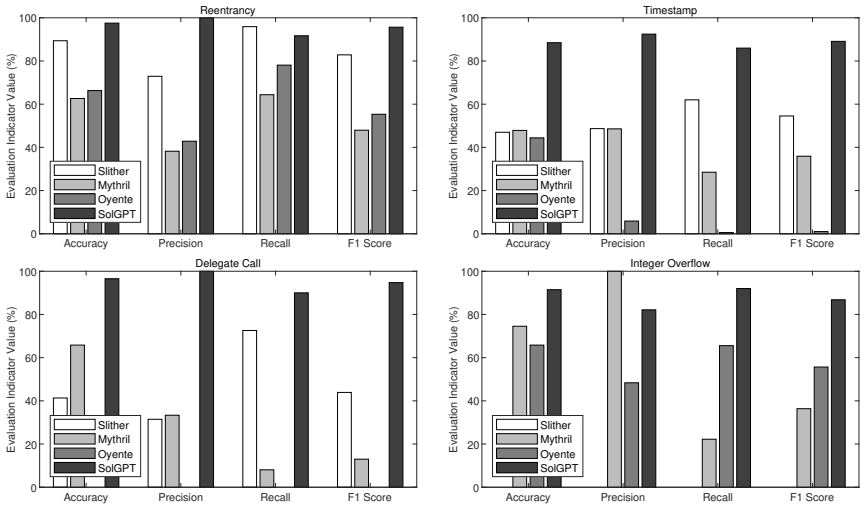


**Fig. 3.** Comparisons with traditional approaches.

Initially, a comparison is conducted between the performance of SolGPT and that of traditional tools, such as Slither[4], Mythril[8], and Oyente[7], followed by an analysis of the obtained experimental data. For an in-depth examination of the results concerning each vulnerability type, the reader is directed to Figure 3. **Reentrancy Vulnerability:** SolGPT achieved an accuracy of 97.53%, outperforming Slither (89.37%), Mythril (62.63%), and Oyente (66.30%). In terms of precision, SolGPT achieved a perfect score of 100.00%, significantly higher than the other tools. The recall and F1 score were also superior, with SolGPT achieving 91.67% and 95.65%, respectively. This represents an improvement of 8.16%

in accuracy and 12.81% in F1 score compared to existing traditional solutions for reentrancy vulnerability detection.

**Timestamp Vulnerability:** For timestamp dependency vulnerabilities, SolGPT demonstrated an accuracy of 88.46%, substantially higher than Slither (46.99%), Mythril (47.85%), and Oyente (44.41%). The precision of SolGPT was 92.45%, while the recall and F1 score were 85.96% and 89.09%, respectively, outperforming the other tools by a wide margin. This marks a notable improvement of 40.61% in accuracy and 34.55% in F1 score compared to existing traditional solutions for timestamp vulnerability detection.

**Delegate Call Vulnerability:** In detecting delegate call vulnerabilities, SolGPT achieved an accuracy of 96.55%, compared to Slither's 41.32% and Mythril's 65.81%. Oyente did not provide results for this vulnerability type. this approach achieved perfect precision (100.00%) and outperformed other tools in terms of recall (90.00%) and F1 score (94.74%). The improvement over existing traditional solutions for delegate call vulnerability detection is substantial, with an increase of 30.74% in accuracy and 50.84% in F1 score.

**Integer Overflow Vulnerability:** For integer overflow vulnerabilities, SolGPT achieved an accuracy of 91.46%, while Mythril and Oyente achieved 74.54% and 65.81%, respectively. Slither did not provide results for this vulnerability type. SolGPT's precision was 82.14%, and its recall and F1 score were 92.00% and 86.79%, respectively, surpassing the other tools. This corresponds to a remarkable improvement of 16.92% in accuracy and 31.13% in F1 score compared to existing traditional solutions for integer overflow vulnerability detection.

This work observed that the three traditional tools generally exhibited low performance across various metrics in detecting the four types of vulnerabilities, especially in the case of Timestamp vulnerabilities. The paper speculates that with the increasing complexity of real-world vulnerabilities, traditional tools may struggle to effectively capture the evolving dependencies and related vulnerability patterns. In the case of Timestamp vulnerabilities, which involve dependencies on the contract's timestamp, more sophisticated analysis and reasoning are required to detect potential issues. This exacerbates the performance decline of traditional tools. Furthermore, our examination of the data reveals that this performance decline also affects SolGPT.

This phenomenon emphasizes the imperative of employing advanced methodologies such as SolGPT to address the escalating complexity of real-world vulnerabilities. SolGPT is designed to encompass the contextual framework within which code is composed. It possesses the capability to discern the interconnections between various components of the code and their interactions, a crucial aspect for the detection of vulnerabilities that might hinge on particular sequences or combinations of operations. As vulnerabilities continue to evolve, the demands for a more nuanced analysis intensify, and SolGPT stands out as an exceptional solution in this respect.

### 5.2 Comparisons with Deep Learning Approaches

In order to further validate the performance of the proposed SolGPT model, it is compared with various deep learning-based schemes such as ReChecker (based on Word2Vec and RNN, LSTM, BiLSTM, BiLSTM-att)[17] and DR-GCN(TMP)[20]. The results are shown in Figure 4.

Based on the data provided in Table 4, SolGPT can analyze the performance of different deep learning schemes in detecting four types of vulnerabilities: Delegatecall Vulnerability, Integer Overflow Vulnerability, Reentrancy Vulnerability, and Timestamp Vulnerability. The deep learning schemes include RNN, LSTM, BiLSTM, BiLSTM-ATT, TMP, and the proposed SolGPT model.

**Table 4.** Comparisons with deep learning approaches.

| Vulnerability Type | Evaluation Metric | RNN | LSTM | BiLSTM | BiLSTM-ATT | TMP | SolGPT |
|---|---|---|---|---|---|---|---|
| Reentrancy Vulnerability | Accuracy | 76.83% | 73.17% | 76.83% | 78.05% | 78.05% | 97.53% |
| | Precision | 80.00% | NaN | 63.64% | 70.00% | 75.00% | 100.00% |
| | Recall | 18.18% | 0.00% | 31.82% | 31.82% | 27.27% | 91.67% |
| | F1 Score | 29.63% | NaN | 42.42% | 43.75% | 40.00% | 95.65% |
| Timestamp Vulnerability | Accuracy | 64.76% | 57.14% | 68.57% | 55.24% | 71.43% | 88.46% |
| | Precision | 63.79% | 59.52% | 71.74% | 57.50% | 68.85% | 92.45% |
| | Recall | 69.81% | 47.17% | 62.26% | 43.40% | 79.25% | 85.96% |
| | F1-score | 66.67% | 52.63% | 66.67% | 49.46% | 73.68% | 89.09% |
| Delegatecall Vulnerability | Accuracy | 69.49% | 67.80% | 76.27% | 69.49% | 76.27% | 96.55% |
| | Precision | 66.67% | NaN | 69.23% | 60.00% | 100.00% | 100.00% |
| | Recall | 10.53% | 0.00% | 47.37% | 15.79% | 26.32% | 90.00% |
| | F1 Score | 18.18% | NaN | 56.25% | 25.00% | 41.67% | 94.74% |
| Integer Overflow Vulnerability | Accuracy | 66.27% | 74.70% | 78.31% | 85.54% | 66.27% | 91.46% |
| | Precision | 50.00% | 57.14% | 66.67% | 76.67% | 50.00% | 82.14% |
| | Recall | 25.00% | 100.00% | 71.43% | 82.14% | 53.57% | 92.00% |
| | F1 Score | 33.33% | 72.73% | 68.97% | 79.31% | 51.72% | 86.79% |

**Reentrancy Vulnerability:** The SolGPT model outshines other deep learning schemes, achieving the highest accuracy (97.53%), precision (100.00%), recall (91.67%), and F1 score (95.65%). In comparison to the second-best approach, our approach exhibits a remarkable improvement of 19.48% in accuracy and 51.9% in F1 score.

**Timestamp Vulnerability:** SolGPT outperforms other approaches, with the highest accuracy (88.46%), precision (92.45%), recall (85.96%), and F1 score (89.09%). In contrast to the second-best approach, TMP, our approach demonstrates a substantial improvement of 17.03% in accuracy and 15.41% in F1 score.

**Delegatecall Vulnerability:** Comparing the deep learning schemes, SolGPT outperforms other approaches, achieving the highest accuracy (96.55%), precision (100.00%), recall (90.00%), and F1 score (94.74%). In comparison to the second-best approach, BiLSTM, our approach shows a significant improvement of 20.28% in accuracy and 38.49% in F1 score.

**Integer Overflow Vulnerability:** SolGPT performs the best with an accuracy of 91.46%, precision of 82.14%, recall of 92.00%, and F1 score of 86.79%. When compared to the closest competitor, BiLSTM-ATT, our approach demonstrates a notable improvement of 5.92% in accuracy and 7.48% in F1 score.

In the comparative analysis of SolGPT against other deep learning approaches for smart contract vulnerability detection, several notable findings emerged. These findings can be attributed to various factors and provide insights into the strengths and limitations of SolGPT in this specific context.

Firstly, SolGPT demonstrates a substantial improvement in accuracy for three out of the four analyzed vulnerabilities, achieving nearly a 20% boost in accuracy when compared to the other methods. However, when examining the Integer Overflow vulnerability, the improvement in accuracy is comparatively modest, standing at 5.92%. This discrepancy can be elucidated by considering the inherent simplicity of the logic underlying the Integer Overflow vulnerability in contrast to the other three vulnerabilities. This simplicity may lead to higher detection rates by various tools, diminishing the relative improvement achievable by SolGPT in this case.

Moreover, this study unveils a remarkable enhancement in the Recall metric of SolGPT for detecting Reentrancy and Delegatecall vulnerabilities in comparison to the optimal solution. The Recall improvement for Reentrancy and Delegatecall vulnerabilities is 59.85% and 42.63%, respectively. This substantial boost in Recall signifies SolGPT's superior ability to correctly identify positive samples for these two vulnerabilities. This capability is particularly critical in vulnerability detection tasks where false negatives (missed vulnerabilities) can have severe consequences, often outweighing the cost of false positives.

Several factors contribute to these noteworthy results. Firstly, the control flow associated with Reentrancy and Delegatecall vulnerabilities is more intricate and challenging to model than that of other neural networks. SolGPT's extensive pre-training allows it to capture and understand these complex control flows effectively, contributing to its improved Recall. Secondly, the dataset used in the analysis contains a limited number of positive samples for Reentrancy and Delegatecall vulnerabilities. SolGPT's superior generalization capabilities, honed through its pre-training phases, enable it to maintain a high Recall rate even with a scarcity of positive samples. This indicates that SolGPT can leverage its comprehensive understanding of Solidity code, acquired during pre-training, to make accurate predictions even when faced with data scarcity.

In summary, the observed differences in performance between SolGPT and other deep learning approaches in the context of smart contract vulnerability detection can be attributed to the complexity of vulnerabilities, the nature of control flows, and SolGPT's ability to generalize effectively from pre-training. SolGPT's strengths in understanding intricate control flows and handling limited data make it particularly potent in identifying vulnerabilities like Reentrancy and Delegatecall, which are of paramount importance in security-critical applications.

### 5.3 Ablation Studies

In this section, we emphasize the importance of ablation studies as a crucial aspect of our evaluation. Ablation studies involve systematically disabling various components or modules within our proposed model to assess their individual

contributions to the overall performance. By conducting these experiments, we aim to provide a clear understanding of the significance of each component in our approach and how it impacts the results. This analysis allows us to identify key mechanisms, optimize our model's architecture, and guide future developments in smart contract vulnerability detection. Ablation studies serve as a foundation for our performance evaluation, enabling readers to follow our rationale and insights effectively.

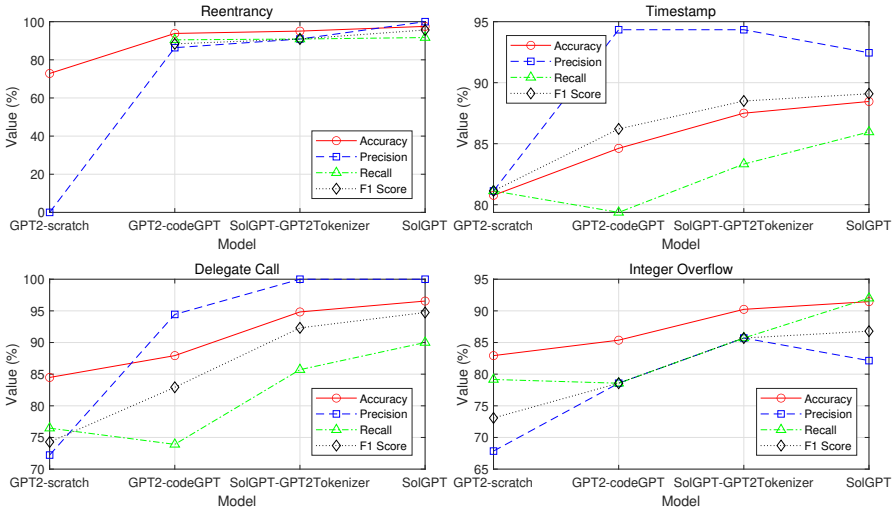


**Fig. 4.** Comparisons with traditional approaches.

In this subsection, we investigate the influence of SolTokinzer and solidity adaptive pre-training on the task of detecting vulnerabilities in smart contracts. A suite of 16 models was trained, specifically for four distinct vulnerabilities in smart contracts, utilizing four distinct upstream models: GPT2-scratch (which involves no pre-training), GPT2-codeGPT (which leverages codeGPT weights as the pre-training initialization point), SolGPT-GPT2Tokenizer (which uses codeGPT weights as a starting point, followed by training with solidity adaptive pre-training), and SolGPT (which is identical to SolGPT-GPT2Tokenizer, but replaces GPT2Tokenizer with SolTokenizer). The hyperparameters and training procedures employed are consistent with those outlined earlier. The comparative performance of these diverse models under the four vulnerabilities is depicted in Figure 4.

Inspection of the results displayed in Figure 4 corroborates the efficacy of the proposed SolGPT model. This is exemplified by the discernible gap between each procedure or module, which provides a clear comparison of their performance.

A clear trend is demonstrated in Figure 4: the performance metrics for the four types of vulnerabilities consistently improve with the integration of the

training steps or modules discussed in this paper. This enhancement is observable when compared to the baseline GPT2-scratch model, which has not undergone any pre-training. Notably, even the performance metrics of the GPT2-scratch model exhibit superiority in the Timestamp and Delegate Call vulnerabilities compared to the best alternative solutions evaluated in this study. The GPT2-scratch model's performance on the remaining two types of vulnerabilities is also highly competitive.

The transition from GPT2-scratch to GPT2-codeGPT demonstrated significant performance enhancement. Leveraging codeGPT as a pre-training initialization point capitalized on the commonalities between Java and Solidity, two programming languages. This allowed the model to learn shared features, leading to noticeable improvements in all performance metrics. On average, across the four types of vulnerabilities, the accuracy increased by a substantial 7.68%.

Moving from GPT2-codeGPT to SolGPT-GPT2Tokenizer, the introduction of solidity adaptive pre-training proved to be pivotal. This additional pre-training task enabled the model to grasp specific language features and keyword nuances unique to Solidity compared to Java. Consequently, the model's performance was further elevated, as reflected by a notable average F1-Score improvement of 5.34% across the four vulnerabilities.

The transition to SolGPT from SolGPT-GPT2Tokenizer marked a refinement in the tokenization process. This optimization ensured the semantic integrity of keywords, effectively emphasizing their semantic information during various training tasks. This nuanced emphasis resulted in a performance boost, as evidenced by an average F1-Score improvement of 2.21% across the four vulnerability types.

These results underscore the progressive impact of incorporating codeGPT-based pre-training and Solidity-specific pre-training on the model's ability to detect vulnerabilities in smart contracts. Additionally, it highlight the importance of optimizing the tokenization process, particularly in preserving and leveraging semantic information related to keywords, to further enhance the model's performance across various vulnerability detection tasks.

While there are a few data points of performance decline in our results, these are rather marginal and mainly concern indices with a specific class label bias, as opposed to the overall accuracy or F1-Score. Further analysis suggests that these declines could be attributed to dataset bias and should not be construed as evidence of overall performance decline.

Collectively, these results suggest that integrating SolTokenizer and solidity adaptive pre-training (SolGPT) substantially improves the model's ability to detect various vulnerabilities in smart contracts. The ablation studies underline the significant contribution of each component to the overall performance of the model. Future efforts should, therefore, focus on further optimizing these components and exploring additional techniques for enhancing the model's performance.

## 6 Conclusion

In today's era of distributed and network-based computing, ensuring the security and dependability of smart contracts has become a paramount concern. As the backbone of blockchain-based systems, smart contracts require robust protection against vulnerabilities that can lead to devastating consequences. To address this critical issue, this paper introduced SolGPT, a specialized GPT model for smart contract vulnerability detection in Solidity. Leveraging Solidity Adaptive Pre-Training, SolGPT boosts feature extraction capabilities while reducing reliance on labeled data. Additionally, SolGPT developed SolTokenizer for accurate and efficient tokenization in the smart contract field. Comparative Experimental Results show that SolGPT outperforms existing models in accuracy, F1 score, and other performance metrics, demonstrating its potential to improve smart contract security. In addition, we conducted ablation experiments to aid readers in gaining a better understanding of the impact of different modules/training methods on the final performance of SolGPT.

While this study constitutes a significant leap forward in enhancing smart contract security, it is not devoid of challenges. A prominent one stems from the inherent limitations in token length for large-scale contract processing in Large Language Models (LLMs) like GPT, used herein as GPT-2 base with a maximum token capacity of 512. To mitigate this, our methodology employs automated control flow analysis to extract vulnerability-related code segments, with the utilized dataset having already undergone this preprocessing step.

The controlled flow chains for the four types of vulnerabilities investigated in this study are not excessively long, hence minimizing the chances of surpassing the token limit. Nonetheless, an acknowledgment is in order that for vulnerabilities with inherently long control flow chains, the extracted code segments may still exceed the token limit, leading to truncation of the segment. This truncation could potentially impair semantic extraction and subsequent detection accuracy, a drawback warranting further investigation.

As we chart the course for future endeavors, our research focus includes refining the pre-training and fine-tuning processes of SolGPT to better encapsulate these challenges and to ensure more robust semantic extraction even with long control flow chains. Moreover, we aim to delve deeper into the implementation of advanced deep-learning techniques, designed to elevate the security of smart contracts further.

These endeavors are not merely academic pursuits but are essential to foster a more secure and reliable smart contract environment, an area of increasing importance in the rapidly advancing digital and decentralized world. We hope that our ongoing research will pave the way for innovative solutions, capable of overcoming the challenges faced by the smart contract security field.

## References

1. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized business review p. 21260 (2008)

2. Szabo, N.: Smart contracts: building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought,(16) **18**(2), 28 (1996)
3. Kushwaha, S.S., Joshi, S., Singh, D., Kaur, M., Lee, H.N.: Systematic review of security vulnerabilities in ethereum blockchain smart contract. IEEE Access **10**, 6605–6621 (2022)
4. Feist, J., Grieco, G., Groce, A.: Slither: a static analysis framework for smart contracts. In: 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). pp. 8–15. IEEE (2019)
5. Grishchenko, I., Maffei, M., Schneidewind, C.: A semantic framework for the security analysis of ethereum smart contracts. In: Principles of Security and Trust: 7th International Conference, POST 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings 7. pp. 243–269. Springer (2018)
6. Amani, S., Bégel, M., Bortin, M., Staples, M.: Towards verifying ethereum smart contract bytecode in isabelle/hol. In: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 66–77 (2018)
7. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 254–269 (2016)
8. Mueller, B.: Introducing mythril: A framework for bug hunting on the ethereum blockchain. Online], https://medium.com/hackernoon/introducing-mythril-a-framework-for-bug-hunting-on-the-ethereumblockchain-9dc5588f82f6. Last accessed **6**(03) (2020)
9. Jiang, B., Liu, Y., Chan, W.K.: Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 259–269 (2018)
10. He, J., Balunović, M., Ambroladze, N., Tsankov, P., Vechev, M.: Learning to fuzz from symbolic execution with application to smart contracts. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 531–548 (2019)
11. Lin, G., Xiao, W., Zhang, J., Xiang, Y.: Deep learning-based vulnerable function detection: A benchmark. In: Information and Communications Security: 21st International Conference, ICICS 2019, Beijing, China, December 15–17, 2019, Revised Selected Papers 21. pp. 219–232. Springer (2020)
12. Liu, Y., Tantithamthavorn, C., Li, L., Liu, Y.: Deep learning for android malware defenses: a systematic literature review. ACM Journal of the ACM (JACM) (2022)
13. Guo, N., Li, X., Yin, H., Gao, Y.: Vulhunter: An automated vulnerability detection system based on deep learning and bytecode. In: Information and Communications Security: 21st International Conference, ICICS 2019, Beijing, China, December 15–17, 2019, Revised Selected Papers 21. pp. 199–218. Springer (2020)
14. Ghazal, T.: Data fusion-based machine learning architecture for intrusion detection. Computers, Materials & Continua **70**(2), 3399–3413 (2022)
15. Dong, Y., Chen, X., Shen, L., Wang, D.: Privacy-preserving distributed machine learning based on secret sharing. In: Information and Communications Security: 21st International Conference, ICICS 2019, Beijing, China, December 15–17, 2019, Revised Selected Papers 21. pp. 684–702. Springer (2020)
16. Tann, W.J.W., Han, X.J., Gupta, S.S., Ong, Y.S.: Towards safer smart contracts: A sequence learning approach to detecting security threats. arXiv preprint arXiv:1811.06632 (2018)

17. Qian, P., Liu, Z., He, Q., Zimmermann, R., Wang, X.: Towards automated reentrancy detection for smart contracts based on sequential models. IEEE Access **8**, 19685–19695 (2020)
18. Wang, W., Song, J., Xu, G., Li, Y., Wang, H., Su, C.: Contractward: Automated vulnerability detection models for ethereum smart contracts. IEEE Transactions on Network Science and Engineering **8**(2), 1133–1144 (2020)
19. Zhuang, Y., Liu, Z., Qian, P., Liu, Q., Wang, X., He, Q.: Smart contract vulnerability detection using graph neural network. In: IJCAI. pp. 3283–3290 (2020)
20. Liu, Z., Qian, P., Wang, X., Zhuang, Y., Qiu, L., Wang, X.: Combining graph neural networks with expert knowledge for smart contract vulnerability detection. IEEE Transactions on Knowledge and Data Engineering (2021)
21. Liu, Y., Tantithamthavorn, C., Li, L., Liu, Y.: Deep learning for android malware defenses: a systematic literature review. ACM Journal of the ACM (JACM) (2022)
22. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)
23. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
24. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
25. Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H., Xu, B.: Attention-based bidirectional long short-term memory networks for relation classification. In: Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers). pp. 207–212 (2016)
26. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
27. Sennrich, R., Haddow, B., Birch, A.: Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909 (2015)
28. Asia, M.R.: Codegpt-small-java-adaptedgpt2 model weights. https://huggingface.co/microsoft/CodeGPT-small-java-adaptedGPT2, date of publication not available, Accessed: 2023-1-2
29. Qian, P.: Smart contract dataset (resource2). https://github.com/Messi-Q/Smart-Contract-Dataset (2022), (Accessed: 2022-12-28)
30. Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., Zhong, Y.: Vuldeepecker: A deep learning-based system for vulnerability detection. arXiv preprint arXiv:1801.01681 (2018)