Contextual Language Model and Transfer Learning for Reentrancy Vulnerability Detection in Smart Contracts

Hong-Bang Le
Information Security Laboratory,
University of Information Technology,
Vietnam National University
Ho Chi Minh city, Vietnam
19520396@gm.uit.edu.vn

Duc-Thang Le
Information Security Laboratory,
University of Information Technology,
Vietnam National University
Ho Chi Minh city, Vietnam
19522198@gm.uit.edu.vn

Doan Minh Trung
Information Security Laboratory,
University of Information Technology,
Vietnam National University
Ho Chi Minh city, Vietnam
trungdm@uit.edu.vn

Tuan-Dung Tran
Information Security Laboratory,
University of Information Technology,
Vietnam National University
Ho Chi Minh city, Vietnam
dungtrt@uit.edu.vn

Phan The Duy
Information Security Laboratory,
University of Information Technology,
Vietnam National University
Ho Chi Minh city, Vietnam
duypt@uit.edu.vn

Van-Hau Pham
Information Security Laboratory,
University of Information Technology,
Vietnam National University
Ho Chi Minh city, Vietnam
haupv@uit.edu.vn

ABSTRACT

The proliferation of smart contracts on blockchain technology has led to several security vulnerabilities, causing significant financial losses and instability in the contract layer. Existing machine learning-based static analysis tools have limited detection accuracy, even for known vulnerabilities. In this study, we propose a novel deep learning-based model combined with attention mechanisms for identifying security vulnerabilities in smart contracts. Our experiments on two large datasets (SmartBugs Wild and Slither Audited Smart Contracts) demonstrate that our approach successfully achieves a 90% detection accuracy in identifying smart contract reentrancy attacks (e.g. performing better than other existing state-of-the-art deep learning-based approaches). In addition, this work also establishes the practical application of deep learning-based technology in smart contract reentrancy vulnerability detection, which can promote future research in this domain.

CCS CONCEPTS

• **Security and privacy** → *Distributed systems security*; • **Software and its engineering** → *Software safety*.

KEYWORDS

Deep learning, Language model, Blockchain, Smart contract, Reentrancy vulnerability

ACM Reference Format:
Hong-Bang Le, Duc-Thang Le, Doan Minh Trung, Tuan-Dung Tran, Phan The Duy, and Van-Hau Pham. 2023. Contextual Language Model and Transfer Learning for Reentrancy Vulnerability Detection in Smart Contracts. In *Proceedings of The 12th International Symposium on Information and Communication Technology (SoICT'23).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

SoICT'23, December 07–08, 2023, Hochiminh city, Vietnam

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The rise of blockchain technologies has been accompanied by a surge in popularity and widespread acceptance across various industries [1]. Among the notable developments in this field, smart contracts have emerged as a focal point of attention. By leveraging cryptographic and other robust security mechanisms, a smart contract can be executed accurately by a network of nodes that inherently lack trust in each other, eliminating the need for a third-party intermediary [2].

Ethereum [3] is widely recognized as a prominent public blockchain platform designed for executing smart contracts. With an extensive market capitalization, Ethereum encompasses a substantial value in Ether, its native cryptocurrency. The nature of smart contracts, characterized by their immutability and irreversibility, introduces considerable challenges when addressing security incidents and vulnerabilities. Using programming languages such as Solidity, which is prone to errors, amplifies the potential for exploitable programming flaws within smart contracts. One notable incident occurred in June 2016 when a hacker exploited a vulnerability in the *fallback* function of the DAO, resulting in the theft of 3.6 million Ether [4]. Furthermore, another incident involved a Github user named "devops199" who unintentionally locked multiple digital wallets, causing the loss of approximately 150 million USD worth of Ether [5]. These incidents underscore the critical need for robust smart contract security measures. Consequently, developing practical vulnerability detection tools for smart contracts becomes an urgent requirement, given their integral role in various blockchain platforms and the increasing reliance on smart contract-based transactions.

To date, recent research has devised numerous approaches to identify vulnerabilities in smart contracts, primarily falling into two categories: (1) expert rule-based methods, which encompass static analysis [6, 7], dynamic execution analysis [8], and hybrid approaches combining both static and dynamic analysis [9] and (2)

data-driven approaches, particularly those utilizing deep learning (DL) techniques [10, 11].

There has been a growing interest in DL vulnerability detection as an alternative to traditional expert rule-based methods. P. Qian et al. [11] introduced an innovative approach called BLSTM-ATT, which stands for Bidirectional Long Short-Term Memory with Attention Mechanism. This novel approach is designed to sequentially process source code and assemble contract snippets to detect reentrancy attacks. ESCORT [10] is the first deep neural network-based vulnerability detection system for Ethereum smart contracts that provides lightweight transfer learning on unknown security flaws. Wu et al. [12] introduced a novel approach, Peculiar, which leverages the pre-training model GraphCodeBERT to enhance the detection of vulnerabilities in smart contracts. While deep learning has gained increasing prominence in various domains, applying deep learning methods to smart contract vulnerability detection remains relatively unexplored. This limited exploration can be attributed to the novelty and intricacy inherent in smart contracts.

Motivated by this problem, we construct a representation model for smart contracts that autonomously incorporates the language's syntactic and semantic information. This is followed by leveraging this model as a foundation for developing vulnerability detection models using established neural architectures. This study applies natural language processing (NLP) models based on DL and Transformer architecture to detect reentrancy bugs in smart contracts. We introduce a custom tokenizer based on WordPiece algorithm and pre-defined solidity tokens. The new tokenizer is trained on a smart contract dataset, helping in rapidly separating source code into tokens while preserving its original syntactic structure. The pre-trained model is built via Masked Language Models (MLM). Then, to fine-tune vulnerability detection models, we integrate pre-treated models with either a Multi-layer Perceptron (MLP) or a Long Short-Term Memory (LSTM).

The following is a summary of our contributions:

- We provide a custom tokenizer trained from a smart contract dataset using the WordPiece algorithm and employ pre-defined solidity tokens to provide improved code encodings while preserving the syntactical structure of source code.
- We present two DL models, including customized DistilBERT-MLP and DistilBERT-LSTM, to detect reentrancy vulnerability in smart contracts.
- Our model performance is evaluated and compared on the Slither Audited Smart Contracts and SmartBugs Wild datasets. The experimental results show that our proposed model achieves 90% accuracy and performs better than other state-of-the-art DL-based approaches.

The remainder of the paper is as follows. Section II provides the necessary basic knowledge on smart contracts, details on DL, attention mechanisms, and related works. Next, we present our proposed approach in Section III. After that, we conduct experimental evaluations in Section IV. Finally, the discussion and future development orientation of the topic are both presented in Section V.

2 RELATED WORKS

The LSTM network is a special architecture of Recurrent Neural Networks (RNNs) capable of learning long-term dependencies to address the inability of RNNs to handle data in parallel. This architecture has been popular and widely used until now. In NLP, there is often a need to focus on specific parts of the input sequence, which is why the attention mechanism is utilized. This mechanism enables the model to attend to particular sentences selectively. The Transformer architecture, introduced in [13], consists of the encoder and the decoder. The encoder is composed of six stacked layers, with each layer containing two sub-layers. The first sub-layer is the multi-head self-attention, while the second sub-layer comprises the fully-connected feed-forward layers. BERT and DistilBERT only utilize the encoder part of the transformer model. These encoder-only models prove to be valuable for tasks that require knowledge of the input, such as sentence categorization.

Tann et al. [14] used sequence learning to detect reentrancy vulnerability in the opcode of smart contracts. Particularly, this paper uses one-hot encoding and an embedding matrix to represent the opcode of smart contracts. The obtained code vectors are used as input to train an LSTM model for determining whether the given smart contract is safe or vulnerable (i.e., binary classification). Sun et al. [15] proposed a method based on the Convolutional Neural Network (CNN) model combined with a self-attention mechanism for smart contract vulnerability detection. Firstly, they possess a remarkable generalization capability, allowing them to learn complex structural features from the training data and effectively capture intricate patterns. Secondly, CNNs are specifically designed to process two-dimensional data, making them suitable for analyzing the structured representation of smart contract code. Additionally, CNNs excel in local feature detection through convolutional layers, enabling them to focus on critical segments of the contract and effectively identify vulnerabilities.

However, it is essential to acknowledge some limitations associated with using CNNs in this context. CNNs typically require fixed-size inputs, which may pose challenges when dealing with smart contracts of varying lengths. Preprocessing steps are necessary to transform the contracts into suitable input sizes. Furthermore, CNNs do not retain sequence information, which can be crucial for understanding the order of statements and the overall contract structure. Lastly, a significant amount of training data is often required for CNNs to learn complex features and avoid overfitting, which can be a considerable task in collecting and preparing a sufficiently large training dataset. In conclusion, using CNNs in the above paper offers several advantages, including their ability to capture structural features and effectively detect vulnerabilities in smart contracts. Thus, consideration must be given to the fixed input size requirement, the loss of sequence information, and the demand for a substantial amount of training data when employing CNNs in this domain.

Qian et al. [11] proposed a novel approach for automated reentrancy detection in smart contracts using sequential models. The authors address the issue of reentrancy vulnerabilities, which can lead to serious security breaches in blockchain-based systems. Sequential models, such as RNNs and LSTM networks, offer several

advantages in detecting reentrancy vulnerabilities. Firstly, sequential models can capture the temporal dependencies and order of operations in smart contracts, enabling them to identify potential reentrancy scenarios that involve recursive function calls. This provides a more comprehensive analysis compared to static analysis techniques. Secondly, sequential models can process variable-length input sequences, accommodating smart contracts' dynamic nature and varying lengths.

However, there are certain limitations when applying sequential models for reentrancy detection. One challenge is the need for substantial training data to train the models effectively. Collecting a diverse and representative dataset of smart contracts with reentrancy vulnerabilities can be time-consuming and labor-intensive. Additionally, the models may encounter difficulties capturing more complex reentrancy patterns or detecting subtle variations of reentrancy attacks that evolve over time. In conclusion, using sequential models in the paper presents a promising approach for automated reentrancy detection in smart contracts. By leveraging the temporal dependencies and order of operations, these models offer an improved understanding of reentrancy vulnerabilities. Therefore, the challenges of acquiring training data and capturing complex reentrancy patterns should be considered when employing sequential models for reentrancy detection.

Bidirectional Encoder Representations from Transformers (BERT) [16] consists of a multi-layered Transformer encoder, and has been highly successful in many natural language processing areas. In particular, many researchers have used BERT for analyzing source code [17]. Furthermore, S. Jeon et al. [18] use BERT to increase the accuracy rate for reentrancy vulnerability detection in smart contracts. However, BERT still has disadvantages such as large size and high computational resources, slow speed, limited traditional capabilities, and inability to process new words. Thanks to that, DistilBERT [19] has shown its advantages as a small, fast, cheap, and light Transformer model based on the BERT architecture. Knowledge distillation is performed during the pre-training phase to reduce the size of a BERT model by 40%. With BERT, we can improve the reentrancy vulnerability prediction accuracy rate. Based on the advantages of DistilBERT over BERT, we decided to use encoder models (DistilBERT) to implement the custom reentrancy vulnerability detection model for smart contracts.

3 METHODOLOGY

3.1 Overview

The model is divided into three stages: a tokenization technique that tokenizes code using a custom vocabulary; a pre-training session that creates a code representation model; and a fine-tuning session that adjusts the model to focus on a specific smart contract classifying vulnerability. Fig. 1 shows the 4 main steps of the training process of our model.

3.2 Tokenization pipeline

Tokenization is one of the most critical steps in text preprocessing. This step cannot be skipped when working with traditional NLP or advanced DL techniques. Tokenization, in its simplest form, is dividing a phrase, sentence, paragraph, or one or more text documents into smaller components. Each of these smaller components is called a token. A tokenizer is responsible for preparing the inputs for a model. The library contains tokenizers for all the models.

WordPiece is the tokenization algorithm Google developed to pre-train DistiBERT. It is similar to BPE. We add a new solidity token and train a new tokenizer from DistilBERT tokenizer with our smart contract dataset. Fig. 2 shows our tokenization pipeline. Tokenizer converts smart contract source code into a sequence of tokens and token IDs based on the built-in algorithm and vocabulary. According to the Solidity documentation, we list the collected Solidity tokens as follows:

(1) Operators: *++*, *−−*, *-*, *!*, , etc.
(2) ABI encoding and decoding functions: *abi.decode*, *abi.encode*, *abi.encodePacked*, *abi.encodeWithSelector*, etc.
(3) Block and transaction properties: *blockhash*, *block.basefee*, *block.chainid*, *block.coinbase*, etc.
(4) Mathematical and cryptographic functions: *keccak256*, *sha256*, *ripemd160*, *ecrecover*, *addmod*, *mulmod*, etc.
(5) Other keywords: *int*, *uint*, *bytes*, *address*, *pure*, *view*, *payable*, *public*, *internal*, *revert*, *assert*, etc.

3.3 Pre-training stage

In this stage, we train a standard DistilBERT-based model with an MLM target to learn an informative generic representation of smart contract source code to create pre-trained models. MLM predicts a masked token in a sequence, and the model can attend to tokens bidirectionally. This means the model has full access to the tokens on the left and right. MLM is great for tasks that require a good contextual understanding of an entire sequence. Following the DistilBERT-based model, we set the embedding size to 768 dimensions. It is the basic embedded knowledge of the model, which will be essential for further fine-tuning tasks.

3.4 Fine-tuning stage

Transfer learning is a foundational concept in AI and ML, involving the application of a pre-trained model to a novel yet related task. This process entails taking an existing pre-trained model, such as BERT or GPT, originally trained for a specific task (the source task), and adapting it to excel in a distinct but related task (the target task).

Fine-tuning is a specialized approach within the broader domain of transfer learning. It encompasses the process of adapting a pre-trained model, typically trained on a larger dataset for a related task, to a specific task by further training it using a smaller, task-specific dataset. Fine-tuning involves the procedure of refining a pre-trained model to excel in a particular task. The primary aim is to enhance performance in the target task by harnessing the knowledge embedded within the pre-trained model.

In our fine-tuing stage, we continue the training of the pre-trained model on a particular downstream task, and in our context, this task pertains to the detection of vulnerabilities in smart contracts. We employ two distinct classification methods during the fine-tuning process. The initial approach involves utilizing a conventional multilayer perceptron (MLP) in conjunction with the pre-trained model, while the second approach employs a Long Short-Term Memory (LSTM) network.

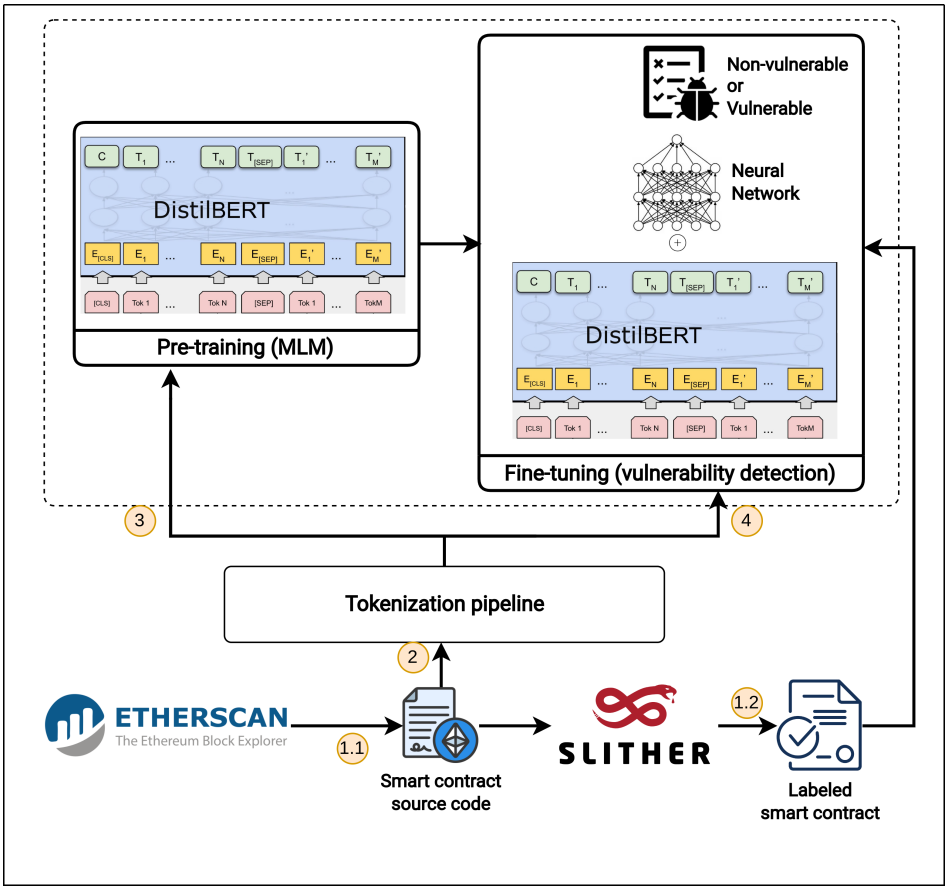

Figure 1: The overall proposed system architecture

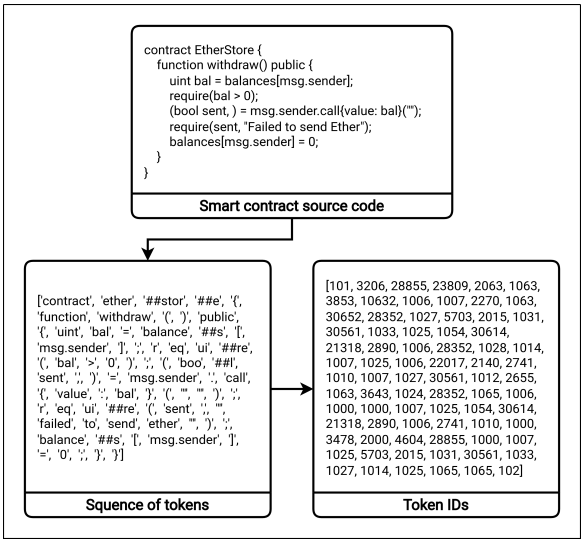

Figure 2: Tokenization pipeline

(1) **DistilBERT-MLP**: For binary classification, we construct a fully connected layer of 768 neurons and one output layer of 1 neuron. Throughout the fine-tuning process, we utilize the pre-trained weights and continue training for multiple epochs. This methodology aligns with the prevalent practice for fine-tuning pre-trained models, capitalizing on the entire pre-trained architecture with minimal alterations.

(2) **DistilBERT-LSTM**: In our approach, we extract the embedding weights from the pre-trained model and utilize them to create the foundational embedding layer for a Long Short-Term Memory (LSTM) network. This LSTM network serves as an intermediary, capturing intricate sequential patterns within the data, and is subsequently linked to a fully connected layer. Finally, the output layer, tailored for classification, leverages the features extracted from the LSTM to make informed decisions, enhancing the ability to discern intricate patterns within the data for accurate classification.

3.5 The structure of DistilBERT-based detection model

This section presents the structure of DistilBERT-LSTM. The DistilBERT-MLP model shares the same characteristics as the pre-trained DistilBERT model. First, we tokenize (1) Solidity source code using our custom tokenizer.

$$token\_ids, attention\_mask = Tokenizer(source\_code) \quad (1)$$

Then, we embedded source code into a dense vector space using our pre-trained model (2).

$$vector\_space = DistilBERT(token\_ids, attention\_mask) \quad (2)$$

The LSTM layer (3) has 2 layers and 256 hidden units, which takes the output of the DistilBERT model as input. After the LSTM layer, a linear layer (4) maps the concatenated output of the LSTM layer to a 256-dimensional feature vector.

$$lstm\_output = LSTM(768, 256, 2)(vector\_space) \quad (3)$$

$$pre\_classifier = Linear(512, 256)(lstm\_output) \quad (4)$$

This feature vector is then passed through the ReLU activation function and the dropout layer (5) before being passed through the classifier layer (6) to obtain the final output.

$$r = Dropout(p = 0.3)(pre\_classifier) \quad (5)$$

$$classifier\_output = Linear(256, 1)(r) \quad (6)$$

4 EXPERIMENTAL RESULTS & EVALUATIONS

4.1 Experimental settings

Dataset. For the pre-training process, we use 42,000 smart contracts from SmartBugs Wild [20] and approximately 70,000 smart contracts from the Slither Audited Smart Contracts Dataset [21].

For the fine-tuning process, we construct two datasets (i.e. *Test_1* and *Test_2*) and split each of them into a 7:3 ratio for training and testing. *Test_1* comprises a total of 10,268 smart contracts sourced from the Slither Audited Smart Contracts dataset. Within this dataset, there are 7,188 smart contracts in the training set and 3,080 smart contracts in the testing set. On the other hand, *Test_2* consists of 5,742 smart contracts obtained from the SmartBugs dataset. Among these, 4,019 smart contracts are in the training set, while the testing set contains 1,723 smart contracts. The specific details of these datasets are presented in Table 4.1.

Table 1: The description of datasets for fine-tuning process.

Dataset	Training set		Testing set	
	reentrancy	non-reentrancy	reentrancy	non-reentrancy
Slither Audited Smart Contracts (Test_1)	3,392	3,796	1,486	1,050
SmartBugs Wild (Test_2)	2,100	1,920	900	822

Setup. Our proposed model was implemented using transformers, tokenizers, and the PyTorch library, all running in Python. The computer's hardware configuration comprises an 8-core CPU, 32GB of RAM, and 60GB of HDD disk space.

Evaluation metrics. We adopt the widely used metrics that include accuracy, precision, recall, and f1-score.

The experimental process. First, an unlabeled subset of data from the datasets is chosen. This subset of data is utilized for training a new tokenizer that adapts to the DistilBERT model. Then, the new tokenizer is trained using the selected subset of data. The tokenizer breaks down the input text into a sequence of tokens the model can process. The training process involves learning a vocabulary of tokens and their corresponding token embeddings. The vocabulary size is then resized to 52,000. This means the tokenizer will only consider the top 52,000 most frequently occurring tokens in the training data. The pre-training process begins by using the pre-trained DistilBERT model and training it on the labeled train and validation sets using MLM objectives. MLM is a task where the model is presented with a sequence of tokens where some tokens have been masked, and the model is required to predict the missing tokens. This helps the model learn contextualized representations of the input text. Finally, we fine-tune pre-trained models to detect reentrancy vulnerability.

- Fine-tuning with MLP: The pre-trained DistilBERT model is fine-tuned using an MLP for the reentrancy detection task. During fine-tuning, the weights of the DistilBERT model reuse the pre-trained models, while the weights of the MLP are updated using backpropagation regarding the classification loss. The output of the MLP is a probability distribution over the possible classes, which can be used to predict the class label of the input source code.
- Fine-tuning with LSTM: The pre-trained model is first loaded, and all its parameters are frozen. On top of the pre-trained model, we add an LSTM layer, a fully connected layer, and one output layer for classification. The resulting model is then trained on a training set using the binary cross-entropy loss and the Adam optimizer. After training, the model is evaluated on a validation set and fine-tuned as needed. Finally, the model is used to make predictions on a test set.

This work aims to develop a machine-learning model for detecting reentrancy vulnerabilities in Solidity smart contracts. We employ a transfer learning approach that leverages large pre-trained language models fine-tuned through supervised learning techniques.

We used smart contracts extracted from the Hugging Face dataset and SmartBugs Wild to train models. We employed the unlabeled data as the source code for Solidity to build pre-trained models using an MLM. We then fine-tuned these models using labeled data analyzed by the Slither tool. Specifically, we used DistilBERT-based MLP and DistilBERT-based LSTM models for this purpose. We used the binary cross-entropy loss function to improve our models. We also used the Adam optimization algorithm for gradient descent in all our experiments. For our models, we set the learning rate (lr) to 1e-05. To mitigate overfitting, dropout regularization was applied during the fine-tuning of our model architectures. We systematically tuned the dropout rate (dr) through preliminary experiments, identifying a value of 0.3 as optimal for preventing overfitting across

all model configurations. Employing dropout regularization in this manner improved the generalization capability of the models on unseen data by reducing co-adaptation between neurons during training.

Additionally, we standardized the batch size to 32 samples for fine-tuning all model variants. This batching approach allowed for an advantageous trade-off between computational expenses and predictive prowess during training. Leveraging mini-batches of 32 inputs achieved efficient model learning without compromising final performance metrics. The batch size configuration facilitated trained models that could aptly balance computational costs with retained capability for classifying new samples once fine-tuning was completed.

Once we complete the training of the pre-trained model, the fine-tuned models are evaluated on two separate test sets (*Test_1* and *Test_2*) by further partitioning each into distinct training and test splits for evaluation. The reentrancy vulnerability detection model is constructed by the pre-trained model with either MLP or LSTM, which is subsequently trained using the training set mentioned above and evaluated on the testing dataset.

4.2 Experimental Results

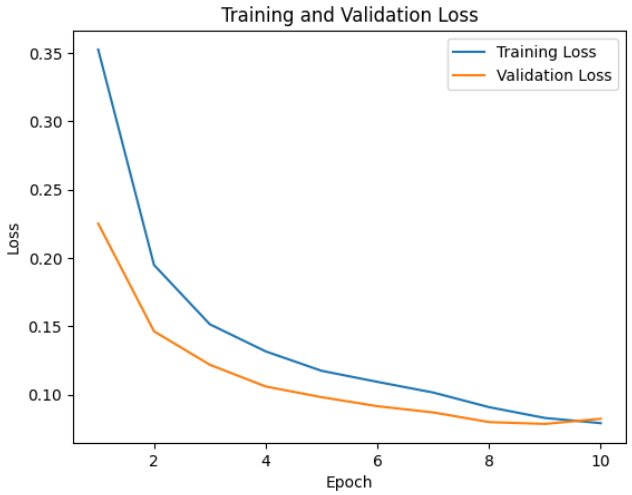

Figure 3: Loss curves on MLM task in SmartBugs Wild Dataset.

The training and validation process plotted in Fig. 3, we report the MLM loss during BERT pretraining on the SmartBugs Wild dataset. As the number of epochs rises, an associated increase or decrease in loss may occur.

We conduct our experiment on two different models, the first is DistiBERT-based MLP. The second model is modified with changes in fine-tuning LSTM for classifications. Table 2 shows the results of two distinct models on two test sets: *Test_1* and *Test_2*.

In *Test_1*, we compare the results of combining custom pre-trained models with MLP or LSTM to detect reentrancy vulnerability in smart contracts. Models Fine-tuning with LSTM show better results. An accuracy of 87.76%, precision of 86.06%, and F1 score of 87.20% indicate that the DistilBERT-based LSTM model is achieving a high level of performance in detecting reentrancy vulnerability in Solidity smart contracts. Accuracy measures the overall correctness of our model's predictions. Hence, an accuracy of 87.76% suggests that our model is correctly predicting reentrancy vulnerability in Solidity smart contracts in most cases. Precision, on the other hand, measures the proportion of true positive predictions among all positive predictions, which indicates that 86.06% of the vulnerabilities predicted by our model are indeed reentrancy vulnerabilities. Finally, the F1 score is a weighted average of precision and recall, providing an overall measure of the performance of a model that balances both metrics.

Table 2: The comparison of performance on two datasets.

Dataset	Models	Acc. (%)	Recall (%)	Pre. (%)	F1. (%)
Test_1	DistilBERT-based MLP	79.05	76.68	78.45	77.55
	DistilBERT-based LSTM	**87.76**	**88.38**	**86.06**	**87.20**
Test_2	DistilBERT-based MLP	81.37	93.08	76.65	84.07
	DistilBERT-based LSTM	**90.19**	**90.33**	**91.03**	**90.68**

In *Test_2*, the results of our DistilBERT-based LSTM models for smart contracts are highly promising, with an overall accuracy of 90.19%. This accuracy indicates that our models correctly classify vulnerable and non-vulnerable smart contracts. Moreover, our models exhibit a commendable recall rate of 90.33%, which signifies their capability to identify a substantial proportion of actual vulnerabilities among smart contracts. The precision of 91.03% is equally noteworthy, demonstrating that when our models classify a smart contract as vulnerable, they are accurate in their assessment. Finally, the F1-score of 90.68% reflects a harmonious balance between precision and recall, indicating that our models are well-rounded in identifying reentrancy vulnerability and avoiding false positives.

After conducting a thorough empirical analysis to assess the effectiveness of our proposed models, our attention now turns to comparing the performance of our detection models with existing DL-based methods for identifying vulnerabilities in smart contracts. We consider two sequential models on two datasets, RSC and RCS [11]. These models are designed to modify the source code of smart contracts and combine contract fragments for reentrancy detection tasks. The authors employ the word2vec technique, which encodes keywords within smart contracts as vectors with predefined dimensions. In the RSC dataset, the models are evaluated using the original smart contracts, while in the RCS dataset, they conduct comprehensive experiments using a dataset composed of contract snippets.

As depicted in Table 3 and Fig. 4, our findings reveal that our DistilBERT-based LSTM model performs slightly better than RSC and RCS across various performance metrics, including accuracy, recall, precision, and f1-score. To elaborate, the DistilBERT-based

Table 3: Performance comparison with existing DL-based approaches on the SmartBugs Wild dataset.

Methods	Models	Acc. (%)	Recall (%)	Pre. (%)	F1. (%)
RSC	LSTM	67.14	55.23	72.50	62.70
	BiLSTM	70.14	61.90	69.14	65.32
	BiLSTM-ATT	73.83	65.48	70.50	67.89
RCS	LSTM	81.73	76.41	88.16	80.06
	BiLSTM	85.38	86.57	85.23	85.55
	BiLSTM-ATT	88.47	88.48	88.50	88.26
Auto-encoder models (DistilBERT)	DistilBERT-based MLP	81.37	93.08	76.65	84.07
	DistilBERT-based LSTM	**90.19**	**90.33**	**91.03**	**90.68**

LSTM model attains an impressive 90.19% accuracy, 91.03% precision, 90.33% recall, and a remarkable 90.68% f1-score.

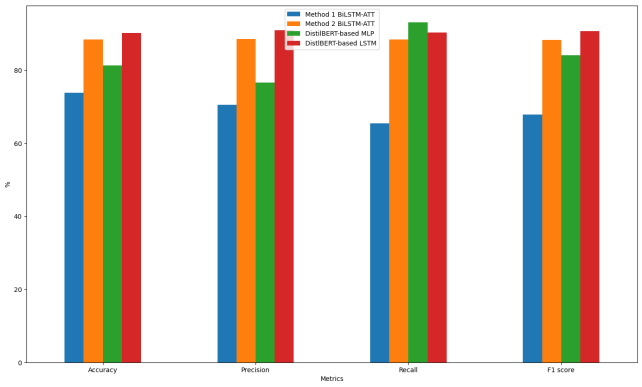

Figure 4: Accuracy, Precision, Recall and F1-score distribution of our method over existing DL-based approaches.

5 CONCLUSIONS

In this work, we provide a customized DistiBERT-based LSTM model for reentrancy vulnerability detection problem in smart contract. Tokenizer is enhanced with the keywords of Solidity and trained on the smart contract dataset, making the token splitting process efficient while respecting the semantics of the programming language. As a result, a custom tokenizer assists pre-trained models in learning a comprehensive representation of Solidity source code. Furthermore, pre-trained models are fine-tuned with LSTM to improve smart contract vulnerability detection. Although we trained our model on reentrancy vulnerabilities, the model can be expanded to additional vulnerabilities such as integer overflow or underflow, time manipulation, etc. As part of our future work, we intend to design a multimodel from the current model DistilBERT combined with a graph model. What is more, our objective is to enhance the efficacy of vulnerability detection in the model by scrutinizing a wider range of information extracted from the source code of the smart contract through the employment of a graph model.

ACKNOWLEDGMENTS

This research was supported by The VNUHCM-University of Information Technology's Scientific Research Support Fund.

REFERENCES

[1] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". In: *Decentralized business review* (2008), p. 21260.
[2] Weiqin Zou et al. "Smart contract development: Challenges and opportunities". In: *IEEE Transactions on Software Engineering* 47.10 (2019), pp. 2084–2106.
[3] Gavin Wood. [n.d.] *Ethereum: A secure decentralised generalised transaction ledger Byzantium Version*. https://ethereum.github.io/yellowpaper/paper.pdf.
[4] *The Dao Attack*. 2016. URL: https://en.wikipedia.org/wiki/The%5C_DAO%5C_(organization).
[5] *Ethereum Parity Bug*. 2017. URL: https://mashable.com/article/ethereum-parity-bug.
[6] Dong Wang, Bo Jiang, and WK Chan. "WANA: Symbolic execution of wasm bytecode for cross-platform smart contract vulnerability detection". In: *arXiv preprint arXiv:2007.15510* (2020).
[7] Yuchiro Chinen et al. "RA: Hunting for re-entrancy attacks in ethereum smart contracts via static analysis". In: *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE. 2020, pp. 327–336.
[8] Noama Fatima Samreen and Manar H Alalfi. "Reentrancy vulnerability identification in ethereum smart contracts". In: *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE. 2020, pp. 22–29.
[9] Noama Fatima Samreen and Manar H Alalfi. "Smartscan: an approach to detect denial of service vulnerability in ethereum smart contracts". In: *2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE. 2021, pp. 17–26.
[10] Christoph Sendner et al. "Smarter Contracts: Detecting Vulnerabilities in Smart Contracts with Deep Transfer Learning." In: *NDSS*. 2023.
[11] Peng Qian et al. "Towards automated reentrancy detection for smart contracts based on sequential models". In: *IEEE Access* 8 (2020), pp. 19685–19695.
[12] Hongjun Wu et al. "Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques". In: *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE. 2021, pp. 378–389.
[13] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).
[14] Wesley Joon-Wie Tann et al. "Towards safer smart contracts: A sequence learning approach to detecting security threats". In: *arXiv preprint arXiv:1811.06632* (2018).
[15] Yuhang Sun and Lize Gu. "Attention-based machine learning model for smart contract vulnerability detection". In: *Journal of Physics: Conference Series*. Vol. 1820. 1. IOP Publishing. 2021, p. 012004.
[16] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *ArXiv* abs/1810.04805 (2019).
[17] Zhangyin Feng et al. "CodeBERT: A Pre-Trained Model for Programming and Natural Languages". In: *ArXiv* abs/2002.08155 (2020).
[18] Sowon Jeon et al. "SmartConDetect: Highly Accurate Smart Contract Code Vulnerability Detection Mechanism using BERT". In: 2021.
[19] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *ArXiv* abs/1910.01108 (2019).
[20] Thomas Durieux et al. "Empirical review of automated analysis tools on 47,587 ethereum smart contracts". In: *Proceedings of the ACM/IEEE 42nd International conference on software engineering*. 2020, pp. 530–541.
[21] Martina Rossini. *Slither Audited Smart Contracts Dataset*. 2022.