Full-Stack Hierarchical Fusion of Static Features for Smart Contracts Vulnerability Detection

Wanqing Jie*, Arthur Sandor Voundi Koe†, Pengfei Huang‡, and Shiwen Zhang§
*†‡Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou 510006, China
§ College of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China
Email: *1254297167@qq.com, †2517482859@qq.com, ‡sawaxyy@gmail.com, §shiwenzhang@hnu.edu.cn
Corresponding Author: †Arthur Sandor Voundi Koe (2517482859@qq.com)

***Abstract*—The security of smart contracts has drawn attention in recent years due to their immutability and ability to hold assets. Existing machine learning and deep learning methods addressing vulnerabilities in smart contracts often partially combine pooled features from first the contract source code, second, the build based approach made of features extracted during source code compilation, and third, the bytecode approach relying on features obtained from the Ethereum virtual machine bytecode analysis. Together those three approaches form the full-stack, and they are usually being conducted under static analysis thanks to its speed of execution. However, to the best of our knowledge, no single work has yet simultaneously undertaken a full-stack intralayer and cross-layer features fusion for smart contracts vulnerability assessment under static analysis, without making use of expert-based patterns nor without manually fusing the various features extracted from shuffled partial combinations of layers in the full-stack. This paper introduces a full-stack hierarchical fusion of static features for smart contracts vulnerability detection. In our construction, we associate each layer of the full-stack to a modality and leverage automatic intramodality and crossmodality pooled features fusion from state-of-the-art artificial neural networks and deep neural networks. Additionally, our models are applied to the hierarchy of power set layers in the full-stack, without any expert-based rule. Furthermore, our work aims to assess the increase in vulnerability detection performance and provide guidance for future research on smart contracts vulnerability detection.**

***Index Terms*—blockchain; smart contract; static analysis; vulnerability; machine learning; deep learning;**

I. Introduction

The increasing popularity and adoption of blockchain technology have led to a myriad of blockchain solutions, and current investments into worldwide blockchain deployments will reach almost 19 billion USD by 2024 [1]. Additionally, to extend on the first proposed blockchain implementation by Satoshi Nakamoto [2], smart contracts originally depicted by Nick Szabo [3] were first being given proof of concept in the Ethereum blockchain platform [4], to bring in a lot of Turing complete functionalities to the blockchain technology. Furthermore, most of the blockchain solutions in use nowadays support smart contracts. Smart contracts are written in various contract-oriented languages (COL) among which the most popular is Solidity. Along with it, smart contracts enabled blockchains do require a computing layer above the commonly encountered storage and communication layers, such as the Ethereum virtual machine (EVM) [4].

Smart contracts' properties of immutability, as well as asset holders, are the main motivating factors in the literature to tackle smart contracts safety issues [5]. Among those safety issues, codification addressing the need of writing an optimized and correct contract [6], and security covering the various vulnerabilities in smart contracts such as the re-entrancy vulnerability [7], are the main topics of interest. Our research aims to address the security aspect of smart contracts while leaving the codification issue out of the scope of this work. As conventional vulnerability detection methods rely heavily on expert-based rules, they exhibit poor scalability and very low accuracy. Deep learning was proposed to address those drawbacks. Furthermore, the work in [8] shows that multimodal machine learning yields higher performance than single modality machine learning.

This paper introduces a full-stack hierarchical fusion of multimodal features extracted under static analysis, for smart contracts vulnerability detection. Different from [9] claiming the black-box nature of models leads to failure to encode useful expert knowledge provided, our work does not make use of any expert rule. In this paper, we apply binary classification based state-of-the-art machine learning and deep learning models to the contract source code, to essentially graph-like features obtained during compilation under the build-based approach, and additionally to features extracted from the Ethereum virtual machine (EVM) bytecode. Moreover, we consider the function as granularity level for analysis. Furthermore, pooled multimodal features extracted from various machine learning models applied to each non-empty subset in the power set of the full-stack layers are hierarchically fused. In our work, we associate each layer of the full-stack to a modality, and we adopt both intra-modality and cross-modality feature vectors fusion.Fig. 1 below depicts an overview of the full-stack structure which exhibits the three modalities used in this paper, namely the source code layer, the build-based layer, the EVM bytecode layer.

Our scheme brings in the following innovations:

- First, we apply the text convolutional neural network (TextCNN) and random forests models over pre-trained Word2vec [10] and bidirectional encoder representations from transformers (BERT) [11] embedding vectors, obtained from applying corresponding word embedding

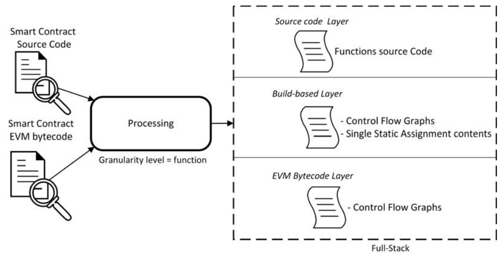


Fig. 1: Overview of the full-stack architecture

models to the function source code, that is extracted using our own built parser. By making use of random forests network, we follow the work of [12] claiming that the best performances in source code based vulnerability detection comes from combining features extracted from both CNNs and tree-based models. We moreover apply a graph convolutional neural network (GCN) to one of the build-based features, namely the various control flow graphs (CFG), each corresponding to a specific function, and we do rely on classical convolutional neural network (CNN) with embedding layer to extract feature vectors from the single static assignment (SSA) form of the function source codeFurthermore, we disassemble the Ethereum virtual machine (EVM) bytecode using the tool Dsol [13] producing graph-like features with a focus on control flow graphs (CFGs) for every identified function in the EVM bytecode, and we employ one GCN to extract feature vectors from the CFGs produced.

- Second, to automatically combine two feature vectors obtained from two different models regarding the same modality, a process denoted as intra-modality fusion, or to automatically combine feature vectors from two different layers or combination of layers, a process known as inter-modality or cross-modality fusion such as in Li et al. [9], and Harer et al. [12] which however focuses on C and C++ vulnerability detection, we make use of two specific kernel functions in the literature. First, we resort to spatial pyramid matching also known as spatial pyramid pooling. Second, we employ cross attention that is explored in [9] for vulnerability detection. Furthermore, fused feature vectors are fed to a bidirectional long short-term memory (Bi-LSTM) with a self-attention mechanism, and we investigate the use of conditional random field (CRF) layer.
- Third, we assess in terms of performance what combination of learning models, what features fusing kernel function among spatial pyramid pooling and cross attention, and finally, what hierarchical combination of layers in the full-stack yield the best results for our hybrid network model. Furthermore, our solution is resilient to whether the contract to be analyzed is only available in EVM bytecodes or not, as not all contracts are published along with their source code on the Ethereum Mainnet.

The remaining of our work is structured as follows. First, Section II reexplores the related works on blockchain technology security and privacy issues, as well as related works on the security of smart contracts. Next, Section III presents background notions needed to grasp the rest of the paper. Then, Section IV presents our system model. Furthermore, Section V details our proposed methodology including data collection, data preprocessing, features extraction, and features fusing. Section VI details some performance analysis results, and finally, Section VII concludes our manuscript and sheds light for future works.

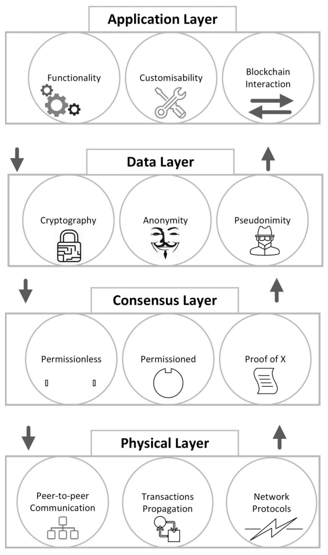


Fig. 2: General blockchain abstraction layer model

II. Related Works

This section revisits two categories of related works. First, the security and privacy issues in blockchain technology. Second, smart contracts vulnerability detection.

A. Blockchain security and privacy issues

Blockchain as a distributed ledger technology still faces some security and privacy issues which can be grouped into four categories, with each category relating to a specific layer of the blockchain abstraction layer model depicted in Fig. 2 below. We note that such a model is not exhaustive and further blockchain abstraction model can be found in [14].

We follow a top-down analysis of the blockchain abstracted architecture in terms of security and privacy layer-wise roles. First, the application layer is responsible for securing the set of blockchain available services from a user point of view, for guaranteeing the safety of contract oriented languages and the conventional execution of smart contracts [15], and finally, for providing a secure and fair incentivizing protocol for honest nodes. Second, the data layer aims at guaranteeing identity, transaction anonymity, and pseudonymity [16], as well as ensuring consistency of the distributed database storage. Third, the consensus layer focuses on enforcing that the honest majority of nodes agree on a decision that will require some price to be paid to maliciously modify the consistency of the world state ledger. However, such price depends on the business application at hand, and therefore on the specific implementation of the consensus mechanism, where many existing consensus protocols can be found in [17], each with their security and privacy issues. Fourth, the communication layer is concerned with network partitioning attacks such as eclipse attacks [18], and on ensuring fair, low latency, and large coverage of transactions propagation in the entire ecosystem.

This paper aims to address the security issues related to the deployment of solidity written smart contracts for the Ethereum virtual machine (EVM).

B. *Smart Contracts Vulnerability Detection*

To cope with smart contracts security issues in the Ethereum blockchain, Argañaraz et al. [19] based on the taxonomy of smart contracts vulnerabilities proposed by Atzei et al. [20], employ static source code analysis to extract functional and security vulnerabilities by formulating some human-expert based verification rules for the target programming language. However, the performance of static analysis depends on the level of abstraction to the source code, and some useful dynamic information cannot be extracted. Gao et al. [21] developed SmartEmbed tool that makes use of source code embedding for clone detection, bug detection, and contract validation. They moreover suggested that relying on expert rules-based systems to detect bugs is a major disadvantage regarding the complexity of the task at hand Furthermore, they advocate for the use of static analysis which is faster, compared to other more reliable but more expensive methods such as dynamic analysis or dynamic symbolic execution. Different from [22] as the closest tool providing similar features, the very interesting work of Slither [23] depicts a static analysis framework for solidity code which makes use of both the contract source code and build-based features extracted from contract source code compilation. However, Slither suffers from too much high-level abstraction to the source code, as well as from a lack of formal semantics [23]. Other tools such as the source code based analyzer SmartCheck [24], and the EVM bytecode based Securify [25], make use of static analysis, but do rely on a limited set of known patterns. Dynamic analyses [26] [27] and dynamic symbolic execution [7] [28] [29], despite their higher reliability over static analysis, are costly to implement and require to execute the smart contract at least once. Furthermore, such an approach may not provide a realistic output if performed in a sandbox or a test network. Moreover, deploying the contract before analyzing it loses the goal of forerunning vulnerabilities detection. Eth2vec [30] is an unsupervised machine learning automatic tool for vulnerabilities and clones detection over solidity written smart contracts. In Eth2vec, contracts with source code are first compiled to extract build-based features, such as the contract abstract syntax tree, as well as the contract assembly code. The additional resulting contract bytecode is then disassembled and processed to extract EVM bytecode-related features. Furthermore, the ground truth label is provided in advance by existing tools. As limitations, however, the work in [30] only detects contracts present in the training dataset, does not perform supervised classification more suitable for natural language processing [31], and limits itself only to the build-based and the bytecode approaches for analysis. Moreover, the internal functioning including the feature vectors fusion at the EVM extractor module is opaque. To extend the coverage of machine learning techniques for smart contracts vulnerability detection, Teng et al. [32] propose a static time-slicing source code analysis approach based on Long short-term memory (LSTM) network to uncover contracts that behave differently from their initial classification on the Ethereum dapps website. However, the authors need to manually extract features to characterize their dataset. Qian et al. [29] use a bidirectional-LSTM with an attention mechanism to focus on Ethereum smart contract reentrancy vulnerability detection at the source code level, and train it over different vector dimensions of word embedding to find the dimension yielding the best performance. Their scheme achieves higher performance over state-of-the-art constructions such as Vanilla-recurrent neural network (RNN), LSTM, and Bidirectional-LSTM. The interesting work of Liu et al. [9] exploits some fixed expert patterns at the source code level, and at the build-based level, then combines both features using an attentive multi-encoder network made of self-attention and cross-attention mechanism, for smart contracts vulnerabilities detection and weights interpretability. However, such a scheme still inherits the weaknesses of relying on expert-based rules, even though it claims that on one hand, only relying on fixed expert rules leads to poor accuracy and poor scalability, and on the other hand, only relying on deep learning models leads to failure to encode the useful expert knowledge due to models black-box nature. The authors, therefore, advocate the use of deep learning over expert rules with interpretability in mind. Moreover, the authors' method does not apply to contracts published without their source code.

To the best of our knowledge, no work has yet been undertaken to simultaneously exploit the source code approach, the bytecode approach focusing on EVM bytecodes, and the build based approach made of features extracted during compilation, in hierarchical combinations, as to provide the best working set of modalities mixtures for enhanced smart contracts vulnerability detection. Moreover, the few works that have attempted such direction required either to input expert-based rules or relied on manual features fusion. The closest literature to our work is the compelling work of Liu et al. [9] which exploits some fixed expert patterns, then combines features extracted from the source code, and the build-based approach, using an attentive multi-encoder network made of self-attention and cross attention mechanism for smart contracts vulnerabilities detection and weights interpretability. However, such a scheme still inherits the weaknesses of relying on expert-based rules and is not applicable over contracts published without their source code.

III. System Model

In this section, we detail the system model employed in this paper, as well as the various machine learning models used in our construction. Moreover, in this work, we first revisit the concept of powerset, where each element of the set is a layer of interest in our architecture. Let S be a finite set of elements where $S = \{s_o, s_1, s_2\}$. The power set or powerset of S, denoted as $\mathcal{P}(S)$, is the set of all subsets of S including S itself as well as the empty set $\emptyset$. The cardinality of $\mathcal{P}(S)$ is defined as $\mathcal{P}(S) = 2^{|S|}$. In our work, we denote the source code layer of our full-stack as SC, the build-based layer as BB, and the EVM bytecode layer as $EVMB$, our full-stack itself being represented as S. Therefore we have $S = \{SC, BB, EVMB\}$, resulting into $\mathcal{P}(S) = \{\{SC\}, \{BB\}, \{EVMB\}, \{SC, BB\}, \{SC, EVMB\}, \{BB, EVMB\}, \{SC, BB, EVMB\}, \{\}\}$.

Regarding our full-stack S depicted in Fig. 1 above, let SC denotes the source code layer, BB denotes the build-based layer, and $EVMB$ denotes the EVM bytecode layer. We consider intra-layers and cross-layers combinations according to the power set of layers in our full-stack, denoted as $\mathcal{P}(S)$, from which we exclude the empty set. Therefore our protocol considers the various combinations of layers of interest to be depicted by each non-empty subset of $\mathcal{P}(S)$, such that $\mathcal{P}(S) - \{\} = \{\{SC\}, \{BB\}, \{EVMB\}, \{SC, BB\}, \{SC, EVMB\}, \{BB, EVMB\}, \{SC, BB, EVMB\}\}$. To each non-empty subset of $\mathcal{P}(S)$, we apply state-of-the-art machine learning models, and obtain pooled features from each subset of $\mathcal{P}(S)$. As detailed in Fig. 3, Fig. 4 and Fig. 5 below, our scheme makes use of both spatial pyramid matching (SPM) also called spatial pyramid pooling (SPP), and of cross-attention kernel functions, to achieve intra-modality and inter-modality feature vectors fusions. We furthermore investigate the impact of adopting conditional random field (CRF) to enhance our models, as positive outcomes related to using CRF have been shown in previous works both on graph convolutional neural networks [33], and on word embedding vectors [34].

A. Source code level

At the SC level, we make use of pre-trained word embedding techniques in the literature, namely word2vec and BERT, over the function source code obtained with our own built parser, and feed each word embedding feature vector to both text convolutional neural network (TextCNN) and random forests (RF) neural network. Then, we realize SC level intra-modality feature vectors fusion between the two models with higher performance among the four as depicted in Fig. 3, using both SPP and cross-attention kernel functions over bidirectional long short-term memory (bi-LSTM) with self-attention. Furthermore, we retain only the intra-modality kernel function providing better feature vectors fusion results between SPP and cross-attention. We moreover investigate the use and effect of conditional random field (CRF) with self-attentive bi-LSTM. Additionally, as said earlier, we adopt functions as our granularity level.

B. Build-based level

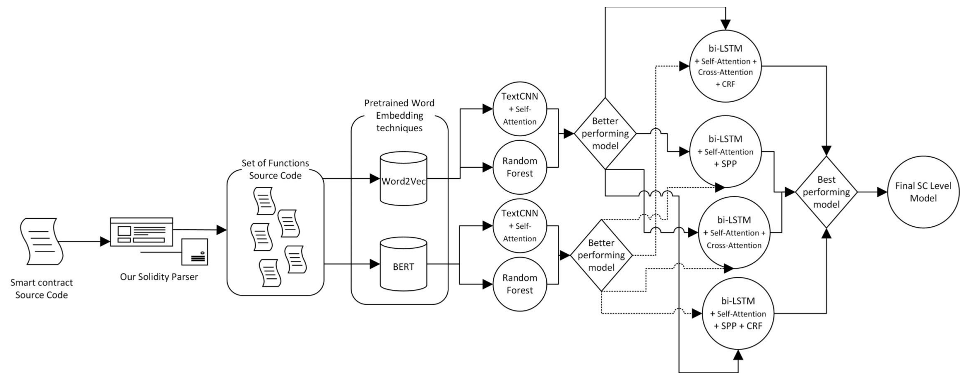
At the BB level, we make use of the slither static analysis tool [23] to extract both the control flow graph (CFG) and the static single assignment (SSA) for each function in the contract source code. Additionally, we make use at this stage of a solidity compiler to produce the EVM bytecode that will be used during the exploitation of the $EVMB$ layer. To extract feature vectors at this layer, first, we make use of a graph convolutional neural network (GCN) applied over the multiple CFGs of the various functions. Since different functions have different CFGs sizes, moreover considering that convolutional layers take inputs of arbitrary sizes while fully-connected layers have a fixed size requirement [35], we investigate the use of both above-mentioned kernel functions, namely SPP and cross-attention over the various CFGs. Then we employ a deep convolutional neural network (CNN) with an embedding layer to extract feature vectors from the single static assignment (SSA) representation of each function. Since we have two internal sub-modalities at the BB level, We furthermore provide the implementation details of our build-based layer (BB) in Fig. 4.

C. Ethereum virtual machine bytecode level

At the $EVMB$ level, we take as inputs the various contracts EVM bytecodes obtained from contract source code compilation using a solidity compiler at the BB level. Then we make use of a solidity decompiler tool in the literature known as Dsol [13] which outputs various CFGs for functions defined within the contract. At this level of our full-stack, we employ a graph convolutional neural network (GCN) that takes as inputs the multiple CFGs from the various functions identified through the existing tool. Due to the difference in size between two functions' CFGs, we investigate both spatial pyramid pooling (SPP) and cross-attention as kernel functions used for feature vectors fusion, to provide a fixed-size input to the fully connected layer which is inherent to all the convolutional neural networks used in this work. The resulting intra-modality fused feature vector is then given as input to a bi-LSTM with self-attention, and we investigate the effect of using a conditional random field layer (CRF). Moreover, we depict the Ethereum virtual machine bytecode layer of the full-stack in Fig. 5.

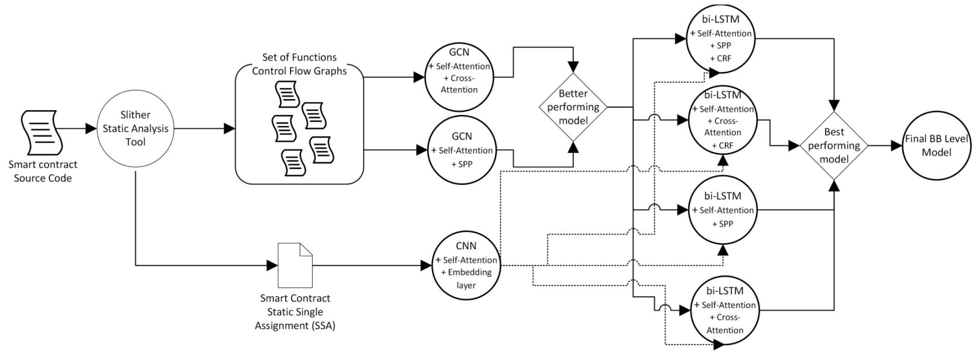
D. Full-stack level

After intra-modality pooled feature vectors fusion, we proceed with the inter-modality feature vectors fusion according to the following non-empty subsets of $\mathcal{P}(S)$ where the order of feature vectors fusion is irrelevant, and where $\mathcal{P}(S) - \{\} = \{\{SC\}, \{BB\}, \{EVMB\}, \{SC, BB\}, \{SC, EVMB\}, \{BB, EVMB\}, \{SC, BB, EVMB\}\}$. First, we combine SC with BB. second, we put together SC with $EVMB$. Third, we associate BB and $EVMB$. Fourth we fuse



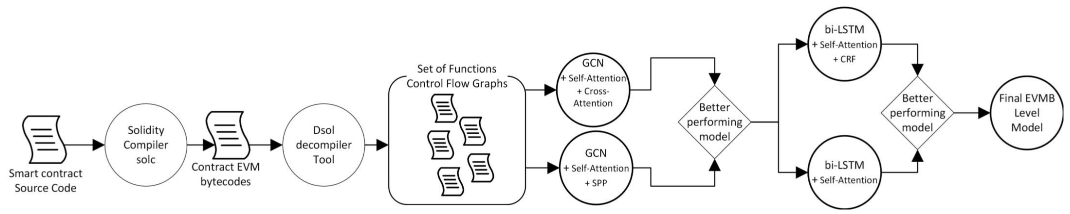
Smart contract Source Code
Our Solidity Parser
Set of Functions Source Code
Pretrained Word Embedding techniques
Word2Vec
BERT
TextCNN + Self-Attention
Random Forest
TextCNN + Self-Attention
Random Forest
Better performing model
Better performing model
bi-LSTM + Self-Attention + Cross-Attention + CRF
bi-LSTM + Self-Attention + SPP
bi-LSTM + Self-Attention + Cross-Attention
bi-LSTM + Self-Attention + SPP + CRF
Best performing model
Final SC Level Model

Fig. 3: Architecture of the source code layer (SC) of the full-stack



Smart contract Source Code
Slither Static Analysis Tool
Set of Functions Control Flow Graphs
GCN + Self-Attention + Cross-Attention
GCN + Self-Attention + SPP
Better performing model
Smart Contract Static Single Assignment (SSA)
CNN + Self-Attention + Embedding layer
bi-LSTM + Self-Attention + SPP + CRF
bi-LSTM + Self-Attention + Cross-Attention + CRF
bi-LSTM + Self-Attention + SPP
bi-LSTM + Self-Attention + Cross-Attention
Best performing model
Final BB Level Model

Fig. 4: Architecture of the build based layer (BB) of the full-stack



Smart contract Source Code
Solidity Compiler solc
Contract EVM bytecodes
Dsol decompiler Tool
Set of Functions Control Flow Graphs
GCN + Self-Attention + Cross-Attention
GCN + Self-Attention + SPP
Better performing model
bi-LSTM + Self-Attention + CRF
bi-LSTM + Self-Attention
Better performing model
Final EVMB Level Model

Fig. 5: Architecture of the EVMB layer of the full-stack

SC, BB, and $EVMB$. At every feature vectors fusion, we investigate the use of the two specific kernel functions deployed in this manuscript, namely spatial pyramid pooling and cross-attention. Moreover, feature vectors fusion is always performed under bi-LSTM with self-attention and we investigate the use of the conditional random field (CRF). In this work, we make use of two levels of attention mechanism at both local intra-modality levels and global inter-modality levels, all represented by non-empty subsets of $\mathcal{P}(S)$, so that all our neural network models are embedded each with a self-attention layer for performances increase, and we always make use of a self-attentive bi-LSTM for inter-modality feature vectors fusion. From this final stage of our experiment, we will be able to conclude what combination of modalities offers the best results in smart contracts vulnerability detection, as well as to acknowledge what specific kernel function to use, between SPP and cross-attention, according to the intra-modality and inter-modality feature vectors fusion scenario at stake, to achieve better detection results. We consider that feature importance addressing model interpretability can be addressed in future work. We summarize in Table I, the hierarchical multimodal fusion of feature vectors in our work according to the non-empty subsets of $\mathcal{P}(S)$ representing the power set of our full-stack S.

IV. Methodology

This section details our methodology including data collection, and data preprocessing.

A. Data Collection

We designed our web crawler over the Ethereum blockchain explorer, and we randomly collected an amount of 20 601 smart contracts files with source code made of 7300 verified contract files and 13 301 verified smart contracts over the

TABLE I: Summary of hierarchical multimodal fusion according to the power set of the full-stack

Intra-modality	**Features**	**Pre-training**	**Neural networks deployed**	**Models used for fusion**	**Kernel functions for fusion**
Source code	functions source code	Word2Vec + BERT	TextCNN + RF	Bi-LSTM ± CRF	SPP + Cross-attention
Build based	CFG + SSA	No	GCN + CNN	Bi-LSTM ± CRF	SPP + Cross-attention
EVM bytecode	CFG	No	GCN	GCN, Bi-LSTM ± CRF	SPP + Cross-attention

Inter-modality	**Set of Neural networks deployed**	**Models used for fusion**	**Kernel functions for fusion**
SC + BB	TextCNN + RF + GCN + CNN	Bi-LSTM ± CRF	SPP + Cross-attention
SC + EVMB	TextCNN + RF + GCN	Bi-LSTM ± CRF	SPP + Cross-attention
BB + EVMB	GCN + CNN	Bi-LSTM± CRF	SPP + Cross-attention
SC + BB + EVMB	TextCNN + RF + GCN + CNN	Bi-LSTM± CRF	SPP + Cross-attention

TABLE II: Summary of strategies used to solve class imbalance issue

$Strategy$	$Embedding$	W_{class0}	W_{class1}	$Accuracy$	$F1$	$Precision$	$Recall$	$AUC - ROC$
None	Bert	1	1	0.9611	0.8386	0.9174	0.7722	0.8831
None	Word2vec	1	1	0.9605	0.8370	0.9239	0.7651	0.8740
INS	Bert	0.1659425	1.7340575	0.9584	0.8249	0.9172	0.7495	0.8679
INS	Word2vec	0.1659425	1.7340575	0.9559	0.8093	0.9307	0.7159	0.8539
ENS	Bert	0.85015092	1.14984908	0.9612	0.8374	0.9267	0.7639	0.8774
ENS	Word2vec	0.85015092	1.14984908	0.9601	0.8338	0.9310	0.7550	0.8727
ISNS	Bert	0.56282351	1.43717649	0.9588	0.8274	0.9164	0.7541	0.8718
ISNS	Word2vec	0.56282351	1.43717649	0.9583	0.8260	0.9263	0.7453	0.8681
SMOTE	**Word2vec**	**1.285251**	**1**	**0.9430**	**0.9452**	**0.9168**	**0.9754**	**0.9427**
SMOTE	**Bert**	**1.285251**	**1**	**0.9416**	**0.9438**	**0.9160**	**0.9734**	**0.9413**

Ethereum Mainnet between November 2020 and April 2021. However, processing those smart contracts resulted in a lot of compilation errors when using either the Slither static analysis tool [23], or the implemented solidity compiler. We believe those compilation errors are due to the differences between the more advanced version of the solidity compiler used on the deployed and verified contracts, and the version of the solidity compiler implemented in our tools. We, therefore, reverted to the openly available Eth2vec smart contracts dataset [30]. The Eth2vec dataset is made of 5000 verified smart contracts published onto the Ethereum Mainnet, such that those contracts can be compiled by solidity version 0.4.11 as compiler. Furthermore, the Eth2vec dataset is made of $95,152$ contracts.

B. Data Preprocessing

Leveraging the various models used in our construction, as well as the multiple hierarchical intra-modality and inter-modality feature vectors fusion in our work, requires extracting features from the contract source code which we consider as our initial state. To process the contract source code, we build a top-down left-to-right with leftmost derivation parser, using the ANTLR tool version 4 [36], based on solidity grammar [37]. Our tool enables parsing the contract source code and extracting the abstract syntax tree, the set of functions, the set of block statements, and the set of simple statements from the contract code. However, we consider as granularity level in this work, the function in a contract code. Our parser coupled with a solidity compiler allows to extract from the contract source code, the assembly code and the EVM bytecodes. We make use of the Slither tool [23] to obtain the ground truth label of our functions as well as to extract build-based features. Moreover, we use the Dsol tool [13] to extract the control flow graph of every identified function in the contract's EVM bytecodes. We manually propagate the function label from the function source code, to the function control flow graph as well as to the function static single assignment. We depict in Fig. 6 below the state transition diagram outlining the various preprocessing steps applied to our data.

Based on our parser, we extract 101,082 functions. The use of Slither static analysis tool for functions labeling results in 87,641 functions classified as non-vulnerable, representing class 0, and 13,441 functions classified as vulnerable therefore representing class 1. We leave feature importance, model interpretability as well as multi-class vulnerabilities classification to future work. To deal with the class imbalance inherent to our data distribution at the source code level, we proceed with the following strategies summarized in Table II while using

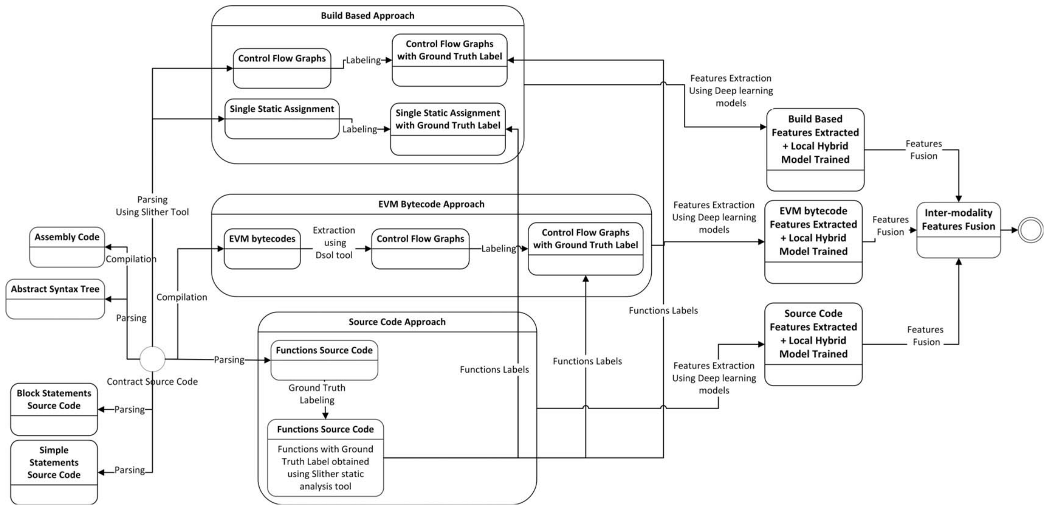

Build Based Approach
Control Flow Graphs
Labeling
Control Flow Graphs with Ground Truth Label
Single Static Assignment
Labeling
Single Static Assignment with Ground Truth Label
Features Extraction Using Deep learning models
Build Based Features Extracted + Local Hybrid Model Trained
Features Fusion
Parsing Using Slither Tool
EVM Bytecode Approach
EVM bytecodes
Extraction using Dsol tool
Control Flow Graphs
Labeling
Control Flow Graphs with Ground Truth Label
Features Extraction Using Deep learning models
EVM bytecode Features Extracted + Local Hybrid Model Trained
Features Fusion
Inter-modality Features Fusion
Assembly Code
Compilation
Abstract Syntax Tree
Compilation
Parsing
Functions Labels
Source Code Approach
Source Code Features Extracted + Local Hybrid Model Trained
Features Fusion
Parsing
Functions Source Code
Functions Labels
Functions Labels
Features Extraction Using Deep learning models
Contract Source Code
Ground Truth Labeling
Block Statements Source Code
Parsing
Functions Source Code
Functions with Ground Truth Label obtained using Slither static analysis tool
Simple Statements Source Code
Parsing

Fig. 6: State transition diagram regarding the various data preprocessing steps

a random forests neural network as our baseline model. For every strategy, we divide our dataset into training and testing sets where the testing set equals 20% of the dataset. The best results are achieved using SMOTE where we undersample class 0 with a factor of 28.5251% and we upsample class 1 to 25 000 samples. We, therefore, extend such a strategy to both our build-based and our EVM bytecode datasets. As for some abbreviations used in Table II, we have the following. The inverse number of samples for INS, effective sample number weighting for ENS, the inverse of the square root of the number of samples for ISNS, and the synthetic minority oversampling technique for SMOTE.

V. Experiments Methodology

For our experiments, we adopt Python as our programming language. We wrote our parser using the ANTLR tool version 4 [36], and installed in a virtual python environment all our required dependencies, among which the Slither tool [23], and the Dsol tool [13]. We adopt scikit-learn (SKLearn) [38] as our artificial neural network library, and will realize our deep learning constructions under Keras and Tensorflow [39]. For our compilers we make use of py-solc-x version 1.1.0, as well as solc-select version 0.2.1. We plan to release our source code over the Github repository.

VI. Conclusion

In this paper, we adopt several state-of-the-art machine learning models over the power set of our full-stack made of the source code layer, the build based layer, and the Ethereum virtual machine (EVM) bytecode layer, and aim to provide the winning combination of layers in the power set, in terms of smart contracts vulnerability detection performance. To achieve our objective, we endeavor to leverage spatial pyramid pooling and cross attention as kernel functions, for intra-modality and inter-modalities feature vectors fusion. We moreover seek to assess the impact of exploiting a conditional random field layer. Extensive design analyses show that our scheme does not make usage of any expert pattern, does not require manual features fusing, and is resilient in case of a missing modality in the testing phase. Furthermore, preliminary data preprocessing has been carried on, and we will release in an extended version our complete experiments, as well as publish our full code over the Github repository. We believe our work is a further step towards enhanced smart contracts vulnerability detection.

References

[1] S. Liu, "Global spending on blockchain solutions 2024 — Statista," 2020. [Online]. Available: https://www.statista.com/statistics/800426/worldwide-blockchain-solutions-spending/
[2] S. Nakamoto and A. P.-t.-p. E. C. System, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://git.dhimmel.com/bitcoin-whitepaper/
[3] N. Szabo, "Smart Contracts : Building Blocks for Digital Markets," 2018.
[4] V. Buterin, "White Paper · ethereum/wiki Wiki · GitHub," p. 1, 2013. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper
[5] M. Bartoletti and L. Pompianu, "An Empirical analysis of smart contracts: Platforms, applications, and design patterns," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10323 LNCS. Springer Verlag, 2017, pp. 494–509. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-70278-0_31
[6] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9604 LNCS. Springer Verlag, 2016, pp. 79–94. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-662-53357-4_6
[7] L. Luu, D. H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the ACM Conference on Computer and Communications Security*, vol. 24-28-October-2016. New York, NY, USA: Association for Computing Machinery, oct 2016, pp. 254–269. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978309
[8] T. Baltrusaitis, C. Ahuja, and L. P. Morency, "Multimodal Machine Learning: A Survey and Taxonomy," pp. 423–443, feb 2019.

[9] 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," vol. 3. International Joint Conferences on Artificial Intelligence, aug 2021, pp. 2751–2759. [Online]. Available: https://github.com/Messi-Q/AMEVulDetector.

[10] L. Q. and M. T., "Distributed Representations of Sentences and Documents," *31st International Conference on Machine Learning, ICML 2014*, vol. 4, pp. 1188–1196, 2014. [Online]. Available: https://dl.acm.org/doi/10.5555/3044805.3045025http://arxiv.org/abs/1405.4053

[11] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, 2019, pp. 4171–4186. [Online]. Available: https://www.aclweb.org/anthology/N19-1423/

[12] J. A. Harer, L. Y. Kim, R. L. Russell, O. Ozdemir, L. R. Kosta, A. Rangamani, L. H. Hamilton, G. I. Centeno, J. R. Key, P. M. Ellingwood, E. Antelman, A. Mackay, M. W. McConley, J. M. Opper, P. Chin, and T. Lazovich, "Automated software vulnerability detection with machine learning," feb 2018. [Online]. Available: http://arxiv.org/abs/1803.04497

[13] Thomas E. Hybel, "GitHub - tehybel/DSol-decompiler: A practical Ethereum smart contract decompiler." [Online]. Available: https://github.com/tehybel/DSol-decompiler

[14] J. Polge, S. Ghatpande, S. Kubler, J. Robert, and Y. Le Traon, "BlockPerf: A Hybrid Blockchain Emulator/Simulator Framework," *IEEE Access*, vol. 9, pp. 107 858–107 872, 2021.

[15] Y. Wang, H. Liu, J. Wang, and S. Wang, "Efficient data interaction of blockchain smart contract with oracle mechanism," 2020, pp. 1000–1003.

[16] W. Cui, X. Song, and Q. Jia, "An Efficient ID-based Fair Off-line Electronic Cash Scheme without Bilinear Pairings," in *2019 4th IEEE International Conference on Big Data Analytics, ICBDA 2019*. Institute of Electrical and Electronics Engineers Inc., may 2019, pp. 286–290.

[17] S. M. H. Bamakan, A. Motavali, and A. Babaei Bondarti, "A survey of blockchain consensus algorithms performance evaluation criteria," *Expert Systems with Applications*, vol. 154, p. 113385, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420302098

[18] M. Kuperberg, "Towards an Analysis of Network Partitioning Prevention for Distributed Ledgers and Blockchains," in *Proceedings - 2020 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2020*. Institute of Electrical and Electronics Engineers Inc., aug 2020, pp. 94–99.

[19] M. C. Argañaraz, M. M. Berón, M. J. Pereira, and P. R. Henriques, "Detection of vulnerabilities in smart contracts specifications in ethereum platforms," in *OpenAccess Series in Informatics*, vol. 83. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, sep 2020.

[20] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts (SoK)," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10204 LNCS. Springer Verlag, 2017, pp. 164–186. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-662-54455-6_8

[21] Z. Gao, L. Jiang, X. Xia, D. Lo, and J. Grundy, "Checking Smart Contracts with Structural Code Embedding," *IEEE Transactions on Software Engineering*, pp. 1–1, feb 2020.

[22] "GitHub - ConsenSys/surya: A set of utilities for exploring Solidity contracts." [Online]. Available: https://github.com/ConsenSys/surya

[23] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proceedings - 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB 2019*. Institute of Electrical and Electronics Engineers Inc., may 2019, pp. 8–15.

[24] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of ethereum smart contracts," in *Proceedings - International Conference on Software Engineering*. IEEE Computer Society, may 2018, pp. 9–16. [Online]. Available: https://orbilu.uni.lu/handle/10993/35862

[25] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the ACM Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, oct 2018, pp. 67–82. [Online]. Available: https://dl.acm.org/doi/10.1145/3243734.3243780

[26] Mythril, "GitHub - ConsenSys/mythril: Security analysis tool for EVM bytecode. Supports smart contracts built for Ethereum, Quorum, Vechain, Roostock, Tron and other EVM-compatible blockchains." 2018. [Online]. Available: https://github.com/ConsenSys/mythril

[27] J. Krupp and C. Rossow, *Open access to the Proceedings of the 27th USENIX Security Symposium is sponsored by USENIX. teether: Gnawing at Ethereum to Automatically Exploit Smart Contracts TEETHER: Gnawing at Ethereum to Automatically Exploit Smart Contracts*, 2018. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/krupp

[28] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *ACM International Conference Proceeding Series*. New York, NY, USA: Association for Computing Machinery, dec 2018, pp. 653–663. [Online]. Available: https://dl.acm.org/doi/10.1145/3274694.3274743

[29] 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.

[30] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, "Eth2Vec: Learning Contract-Wide Code Representations for Vulnerability Detection on Ethereum Smart Contracts," jan 2021. [Online]. Available: http://arxiv.org/abs/2101.02377

[31] F. Hill, K. Cho, and A. Korhonen, "Learning distributed representations of sentences from unlabelled data," in *2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2016 - Proceedings of the Conference*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2016, pp. 1367–1377. [Online]. Available: http://aclweb.org/anthology/N16-1162

[32] T. Hu, X. Liu, T. Chen, X. Zhang, X. Huang, W. Niu, J. Lu, K. Zhou, and Y. Liu, "Transaction-based classification and detection approach for Ethereum smart contract," *Information Processing and Management*, vol. 58, no. 2, p. 102462, mar 2021.

[33] H. Gao, J. Pei, and H. Huang, "Conditional random field enhanced graph convolutional neural networks," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, jul 2019, pp. 276–284. [Online]. Available: https://dl.acm.org/doi/10.1145/3292500.3330888

[34] H. Liu, Z. Zhang, Y. Xu, N. Wang, Y. Huang, Z. Yang, R. Jiang, and H. Chen, "Use of BERT (Bidirectional Encoder Representations from Transformers)-Based Deep Learning Method for Extracting Evidences in Chinese Radiology Reports: Development of a Computer-Aided Liver Cancer Diagnosis Framework," *Journal of Medical Internet Research*, vol. 23, no. 1, jan 2021. [Online]. Available: /pmc/articles/PMC7837998//pmc/articles/PMC7837998/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC7837998/

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, sep 2015.

[36] T. Parr, *The Definitive ANTLR 4 Reference*, 2nd ed. Pragmatic Bookshelf, 2013.

[37] F. Bond, "GitHub - solidityj/solidity-antlr4: Solidity grammar for ANTLR4," 2019. [Online]. Available: https://github.com/solidityj/solidity-antlr4

[38] F. Pedregosa FABIANPEDREGOSA, V. Michel, O. Grisel OLIVIERGRISEL, M. Blondel, P. Prettenhofer, R. Weiss, J. Vanderplas, D. Cournapeau, F. Pedregosa, G. Varoquaux, A. Gramfort, B. Thirion, O. Grisel, V. Dubourg, A. Passos, M. Brucher, M. Perrot andÉdouardand, andÉdouard Duchesnay, and F. Duchesnay EDOUARDDUCHESNAY, "Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot," Tech. Rep., 2011. [Online]. Available: http://scikit-learn.sourceforge.net.

[39] F. J. J. Joseph, S. Nonsiri, and A. Monsakul, "Keras and TensorFlow: A Hands-On Experience," in *EAI/Springer Innovations in Communication and Computing*. Springer Science and Business Media Deutschland GmbH, 2021, pp. 85–111. [Online]. Available: https://doi.org/10.1007/978-3-030-66519-7_4