

Smart contracts vulnerabilities detection using ensemble architecture of graphical attention model distillation and inference network

Preethi¹, Mohammed Mujeer Ulla², Ashwitha Anni¹, Pavithra Narasimha Murthy³, Sapna Renukaradhya¹

¹Department of Information Technology, Manipal Institute of Technology Bengaluru, Manipal Academy of Higher Education, Manipal, India

²School of Computer Science and Engineering & IS, Presidency University, Bengaluru, India

³Department of Computer Science and Engineering, Manipal Institute of Technology Bengaluru, Manipal Academy of Higher Education, Manipal, India

Article Info

Article history:

Received Dec 2, 2023

Revised Aug 17, 2024

Accepted Aug 30, 2024

Keywords:

Blockchain

Ensemble architecture

GAMDI-Net

Security

Smart contracts

ABSTRACT

Smart contracts are automated agreements executed on a blockchain, offering reliability through their immutable and distributed nature. Yet, their unalterable deployment necessitates precise preemptive security checks, as vulnerabilities could lead to substantial financial damages henceforth testing for vulnerabilities is necessary prior to deployment. This paper presents the graphical attention model distillation and inference network (GAMDI-Net), a pioneering methodology that significantly enhances smart contract vulnerability detection. GAMDI-Net introduces a unique graphical learning module that employs attention mechanism networks to transform complex contract code into a smart graphical representation. In addition to this a dual-modality model distillation and mutual modality learning mechanism, GAMDI-Net excels in synthesizing semantic and control flow data to predict absent bytecode embeddings with high accuracy. This methodology not only improves the precision of vulnerability detection but also addresses scalability and efficiency challenges, reinforcing trust in the deployment of secure smart contracts within the blockchain ecosystem.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Sapna Renukaradhya

Department of Information Technology, Manipal Institute of Technology Bengaluru

Manipal Academy of Higher Education

Manipal, India

Email: sapna.r@manipal.edu

1. INTRODUCTION

The inception of smart contracts can be attributed to Nick Szabo [1]. The utilization of electronic methods improves the reliability and efficiency of contract negotiations. Smart contracts enable the efficient execution of secure transactions without the involvement of intermediaries. The proposed concept has received limited attention since its inception, and the absence of a reliable execution environment poses challenges to its implementation in practical applications. The blockchain technology, which was introduced in 2009, has attracted considerable attention from industry experts because of its close association with bitcoin as a foundational component. The term "blockchain" denotes a distributed ledger that exhibits key features such as decentralization, immutability, and operates in a distributed manner. The aforementioned attributes of blockchain technology highlight its capacity to enable diverse applications, including asset securitization, food security, insurance, supply chain financing, and other domains unrelated to virtual currency [2].

Security concerns often arise due to the existence of vulnerabilities in smart contracts. In the given scenario, it is typical to encounter multiple vulnerabilities including reentrancy, access control weaknesses, integer overflows, unchecked function call return values, denial of service, delegate calls, short address attacks, competitive conditions/illegal pre transactions, transaction order dependencies, timestamp dependencies, and numerous others [3]. There are currently several automatic detection methods being developed to tackle the significant challenges that arise from vulnerabilities in smart contracts. The objectives of vulnerability detection and analysis are accomplished by employing automated technologies for vulnerability detection. These technologies facilitate the rapid detection of security vulnerabilities in contracts. The categorization of these vulnerabilities was conducted by the researchers using multiple criteria, including severity, root cause, solidity, security, privacy, performance, ethereum virtual machine (EVM) byte code, and blockchain features [4].

There are two primary classifications of methods employed for identifying vulnerabilities in smart contracts: deep learning-based vulnerability detection methods and traditional method-based vulnerability detection methods. The utilization of expert rules is essential for identifying vulnerabilities in conventional approaches. The guidelines presented in this article are backed by automated vulnerability detection technologies and are derived from empirical knowledge. The popularity of deep learning algorithms has experienced a significant increase in recent years [5]. The prevalence of automated vulnerability detection techniques that integrate symbolic execution with non-symbolic execution approaches is attributed to their exceptional accuracy in identifying well-known vulnerabilities. The utilization of symbolic execution tools in order to simulate symbolic paths necessitates a significant allocation of time resources. The main challenge is to conduct a comprehensive analysis of all possible pathways covered by a contract. Therefore, these technologies are deemed inappropriate for batch vulnerability detection. Non-symbolic execution techniques, such as slither and smart check, rely extensively on predefined detection criteria. It is crucial to acknowledge that there exists a limitation concerning extended simulation durations when dealing with symbolic pathways [6]. The aforementioned circumstances may lead to the possibility of drawing inaccurate and unfavorable conclusions.

The quality of the datasets utilized significantly impacts the performance of deep learning models. The task of incorporating dependable expert information poses a considerable challenge, primarily because of the inherent denseness associated with these models. Moreover, a notable portion of their methodologies demonstrate intricacy. Deep learning algorithms have demonstrated a significant level of accuracy in identifying vulnerabilities, outperforming previous static analysis methods [7]. This task can be achieved without the need for explicitly specifying the detection algorithms that depend on training datasets. Achieving consistent execution durations in deep learning techniques involves the computation of the product between input values generated from the smart contract and the obtained weight values.

Convolutional neural networks (CNNs) have consistently demonstrated a notable degree of accuracy in the detection of malware and vulnerable software codes [8]. CNNs have been widely utilized in deep learning models, primarily for image recognition tasks rather than code analysis. The assessment of semantics and context in smart contracts faces challenges that delay accuracy, leading to a significant occurrence of false positives and false negatives [9]. The utilization of deep learning and machine learning techniques has significantly enhanced the effectiveness and precision in identifying vulnerabilities within ethereum smart contracts (ESC). The scarcity of real-world cases showcasing vulnerabilities in ethereum smart contracts poses challenges for security firms in gathering a sufficient number of instances within a specified timeframe. In order to effectively address challenges associated with identifying vulnerabilities in smart contracts, it is crucial to prioritize research on small-sample learning techniques. The primary goal of this study is to facilitate advancement and accelerate the exploration of solutions within the selected domain [10].

As blockchain technology continues to evolve, smart contracts emerge as a transformative force across various sectors due to their capacity for ensuring trustworthiness, transparency, and automated execution of agreements. Nevertheless, the irrevocable nature of smart contracts once they are deployed to the blockchain heightens the importance of preemptive vulnerability detection. Conventional methods for detecting vulnerabilities, such as manual inspections and static code analysis, are increasingly inadequate due to their limitations in handling complex patterns and extensive datasets. The advent of deep learning, with its exceptional ability to discern complex patterns within vast amounts of data, a new era of potential in identifying and mitigating potential security vulnerabilities in smart contracts. This research endeavors to leverage deep learning to fortify the security of smart contracts, aiming to enhance the detection process's efficiency and scalability, thereby reinforcing the foundational trust that blockchain technology promises.

- Advanced graphical learning module: The proposed graphical attention model distillation and inference network (GAMDI-Net) introduces an innovative learning module that converts smart contract code into graphical representations. This module employs attention mechanism networks to effectively translate both semantic and bytecode information into graphical formats, offering an enhanced understanding of

the underlying structures and relationships within smart contracts, which is a significant advancement over traditional linear representations.

- Dual-modality model distillation: The model distillation approach is introduced, utilizing a network that processes dual-modality inputs during training. This method refines the learning process by distilling complex information into more accessible forms, thereby improving the network's ability to predict vulnerabilities by synthesizing semantic and control flow insights from both source code and bytecode.
- Mutual modality learning mechanism: The research develops a mutual modality learning mechanism that incorporates transfer and mutual loss to facilitate collaborative learning between different network modalities during training. This collaborative approach is pivotal in enhancing the model's capability to infer missing code embeddings and to improve the accuracy of vulnerability detection in smart contracts.
- Innovative inference for vulnerability detection: At the inference stage, the study leverages the distilled model network to predict code embeddings not present in the bytecode. This predictive capacity is essential for increasing the accuracy and reliability of vulnerability detection in smart contracts, presenting a substantial contribution to the field of smart contract security.

2. RELATED WORK

The field has witnessed significant advancements that have enabled the efficient utilization of deep neural networks. As a result, the detection of vulnerabilities in smart contracts has been greatly improved, leading to enhanced efficiency. The positive outcomes derived from the implementation of this practice are illustrated in [11]. Control flow graphs are a commonly used technique in software engineering to visually represent the structure and flow of source code, as exemplified in [12]. The approach described employs long short-term memory (LSTM)-based networks to sequentially parse the source code. The main goal of a sequential model is to analyze the operational code of ethereum [13]. However, these approaches either solely consider the source code or operation code as a text sequence, disregarding semantic blocks, or they fail to take into account crucial aspects of the data flow. Hence, these methodologies produce outcomes as a result of their incapacity to precisely capture the semantic framework.

The speaker verification checker (SVChecker) technique consists of three main stages: code fragment extraction, application of a deep learning model, and the development of a specialized checker for solidity smart contract source code that is not relevant [14]. The methodology outlined here converts the process of detecting vulnerabilities in ethereum smart contracts into a text classification task. The Peculiar [15] system employs pre-training techniques to transform the solidity source code of the ESC into a non-Euclidean graph problem. The utilization of comprehensive data flow graphs improves the detection of vulnerabilities in ESC. The primary dataflow graph displays significant differences compared to the conventional dataflow graphs typically used in modern methodologies.

The development of sequence chart studio (SCStudio) [16] encompassed the establishment of a centralized platform with the objective of enhancing the efficiency of workflows associated with smart contract production. The main goal of this project is to improve the accessibility of secure smart contract development for programmers. The core principle of this approach involves the incorporation of pattern-based learning and security verification via testing, leading to a proficient solution. The proposed methodology provides real-time recommendations for prioritizing the implementation of security upgrades based on their respective levels of importance. The methodology outlined in [17] comprises three distinct phases, the initial phase in the construction of transactions intended for transmission to an agent smart contract involves the generation of fuzzing input. The main purpose of these transactions is to initiate attacks, while also gathering and recording runtime information in the execution log. The next step in the process is to perform preprocessing on the contract in order to improve its detectability. During the contract upload procedure, the source code [18] is subjected to a preprocessing step. The primary objectives of this step encompass two aspects: firstly, the extraction of a candidate pool for fuzzing, and secondly, the identification of any dependencies present in the code. The verification of vulnerabilities is the concluding stage in the process. The primary goal of execution log analysis [19] is to evaluate the vulnerability status of a contract.

The software package consists of a graph extraction method and a highly efficient vulnerability detection tool [20], the next step in the process of extracting a graph involves creating the graph after vulnerability patterns have been extracted. The first step in the development of vulnerability security-centric graph (SCGraph) libraries involves the utilization of the approximation graph matching approach. The vulnerability detection procedure utilizes a methodology to choose sample SCGraphs from the dataset. The following procedure involves calculating the similarity between the SCGraphs produced by the identification-requiring contracts. One essential step that should be given priority is the assessment of the contract's vulnerabilities [21]. The utilization of machine learning techniques [22] in conjunction with ContractWard is highly recommended for the purpose of detecting vulnerabilities in smart contracts. The first step of the method

involves extracting bigram attributes from simplified smart contract operation codes. The model development process integrates two sampling strategies and employs a hybrid approach that incorporates the utilization of five distinct machine learning algorithms. Currently, the testing phase of the ContractWard system involves the execution of 49,502 real-world smart contracts on the ethereum platform, which is further being reviewed.

According to Ma *et al.* [23], conventional methods are commonly utilized to detect vulnerabilities in smart contracts. The selection of methodologies is primarily determined by either static analysis or dynamic execution approaches. Unfortunately, their reliance is heavily dependent on various patterns that have been identified by experts. The presence of human error is an inherent aspect in the manual configuration of patterns. Furthermore, the accurate representation of intricate patterns can present a considerable challenge. The utilization of multiple rigid patterns without sufficient refinement increases the likelihood of encountering false positives and false negatives. Furthermore, proficient adversaries possess the capability to effortlessly bypass pattern checking techniques [24]. Furthermore, the exponential growth of smart contracts presents significant difficulties in identifying patterns, even within a restricted community of specialists. One potential approach to consider is to formally solicit each expert to provide labels for a preselected set of contracts. The training process of a model involves a subsequent procedure that includes the collection and utilization of contracts that have been annotated by multiple experts. The proposed methodology showcases autonomous capability in the detection and precise identification of particular vulnerabilities within a contract [25].

3. PROPOSED METHODOLOGY

This section of the study consists of three major parts, namely ‘A learning module that is represented graphically’, this is used to transform the code of smart contracts to semantic and byte code into graphical representations of control flow by the use of attention method networks, the next phase consists of ‘model distillation’ along with a network having double modality denoted as F as well as for one-modality given as W and lastly a ‘modality learning mechanism that is mutual’, this includes transfer and mutual loss during training. During the training phase, collaboration of the network is learnt, while at inference, the network of model distillation denoted U is used for prediction of code embeddings that are absent in the byte code, this therefore increases the accuracy of detection. It is seen that the model distillation F is operational only in the training stage. Figure 1 shows the embedding on the graphical layer. Figure 2 shows the bidirectional encoder representations from transformers (BERT) model and graph network.

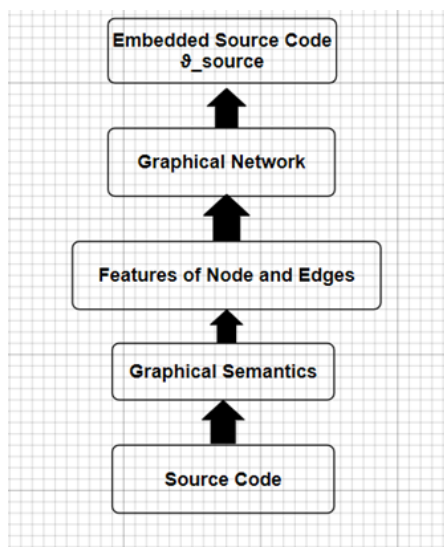


Figure 1. Embedding on the graphical layer

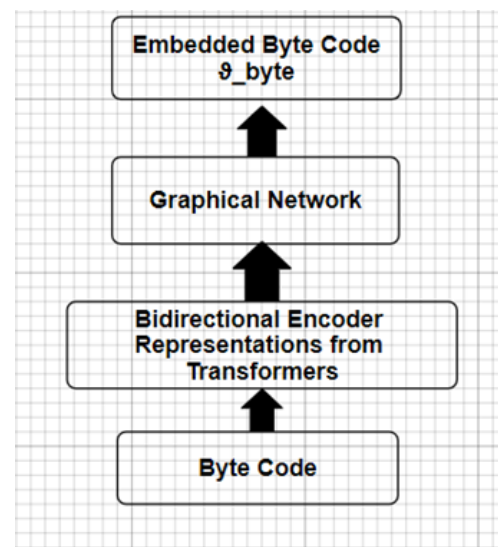


Figure 2. BERT model and graph network

3.1. Learning module based on graphical representation

Previous studies have shown that programs represented graphically are more effective in order to preserve the required structure as well as semantic specifics. Considering this ideology, the byte code as well as the source code of smart contracts are transformed into separate graphs. Similarly, graphical attention networks are used in the processing of the graphs, that facilitate retrieval of essential graphical features.

3.2. Graphical information study

Considering the representation of the source code, a semantic graphical representation is introduced for structure of the dependencies using the code, graphs of two different kinds are retrieved along with three distinct edges. In this case, the nodes are used to represent the different elements of the program that are termed as function call as well as variables, with the flow and control links between nodes that are seized by the edges. Particularly, every edge has a temporal sequence which is constant for the code having a sequence order. In order to extract the flow of the graph, there are ω segments and i flow edge or flow links. There are a set of instructions contained in ω .

- Two phase prior-training: this phase includes two phases of prior training. Considering the instruction stage, masked language job that decodes the segment rules by the prediction of tokens that are masked. In the segment stage, the control flow of connections between linked segments are apprehended by the adjacency segment predicting job.
- Tuning: the vulnerabilities of variables are addressed in this phase, where the tuning that is specific to the vulnerability is utilized. This phase enhances the model to further learn the various kinds of vulnerabilities and therefore improve the accuracy for detection.
- Extraction of features: the network that is tuned is used for the extraction of byte code segment features. By considering the information obtained from the previous two phases, the model is efficient to analyse as well as classify the segment byte codes on the basis of vulnerabilities.
- Graph development: the code for the smart contract data is classified using a code for graphical semantics. Consider the smart contract M for the computation for the possibility of a function m that has a vulnerable re-entrancy. m is initially designed as the main node G_1 since the inside code includes the call data that is summoned. Consider the temporal sequence, the crucial state for variable D which is the sender, as the main node G_2 . Summation, which is the local parameter is designed and developed as a normal node R_1 . To call c , the data is retrieved through the main node as G_3 , as well as the function $h g$ is classified for an attack virtually through the normal node R_j . The nodes as well as the edges have a semantic dependency that is developed in three iterations that is termed as e, f and $h g$. However, every edge depicts the way that is travelled through the testing function and the order of sequence function is shown using the temporal edge.
- The different functions produce graphs using structures as well as graph normalization by omitting the normal nodes as well as combining its features along with the closest main node and transferring its characteristics. The normal nodes are discarded by the aligned edges to preserve the initial or final nodes for transferring the corresponding main nodes. The main nodes are prompted using normalization. Figure 3 shows the processing of graph.
- Evaluation of control flow: the byte code is designed to develop the graphical control flow which is shown in the Figure 3. The code for the smart contract is used for interpretation of the byte code using the compiler as well as the generation of a DEHIG that shows a graphical flow for the byte code that is complied. This includes the basic segment along with control edges.
- Separation of segments: basic segments that are made up of series of virtual machine instructions are divided. The instructions such as RETURN as well as JUMP are towards the end of the segments, that are used for analysing.
- Interlinked control flow: the basic segments that are interlinked via the edges of control flow, depict the links of the segments that are invoked by calls. The edges show the logical links for various program phases.
- Edge classification of control flow: the edges are classified into three:
 - a) Unrestricted jumps: this depicts the direct shifts of control.
 - b) Boolean Condition Jumps: show the branches that are on the basis of conditional results.
 - c) Other possible edges: seize other extra control flow links, that modify the representations graphically.

3.3. Model distillation based on graphs

There are two kinds of graphs, the architecture of the model is developed based on the graph's attention network for perceiving the highest stage semantic embedding for source code θ_{source} as well as byte code θ_{byte} that belongs to V^h . The graph that is embedded retrieves two stages, text transmission as well as load stage. In the stage of text transmission, the information is transferred from the network to the edges accordingly for sequential understanding of code. For every step-in time o the information passes via an ordered sequence. At time step o , the data is transferred to the o – th temporal phase i_o with the hidden phase updated at the final state for every node i_o . Every node's hidden phase is evaluated by assigning it to the neighbours as given (1). For the equation given in (1), the activation function is denoted as γ , R_m shows the nodes next to m graphically, A depicts the matrix α_{mn} which shows the coefficients of attention as given in the (2). Considering the (2), the integration of the terms is shown by the operator \oplus , rectified linear unit (ReLU) function is given

as ∂ and vector of weight for an individual multilayer perceptron (MLP) layer is given as $e \rightarrow \cdot$. after traversal of edges, the resulting graph shows ϑ belongs to V^h by using hidden phases for every node inside the graph. In the (3), the product of elements is shown using \odot and activation function is given as β . Q_n is the matrix shown and the vector bias is shown as f_n with subscript n belongs to $\{1,2\}$ so that it is a network attribute that can be trained. The count of nodes is represented by Z and MLP is given by T .

$$l_m' = (\sum_{n \in R_m} \alpha_{mn} A l_n) \gamma \quad (1)$$

$$\alpha_{mn} = \frac{\exp(\partial(e \rightarrow X[A l_o \oplus A l_n]))}{\sum_{o \in R_m} \exp(\partial(e \rightarrow X[A l_o \oplus A l_n]))} \quad (2)$$

$$\beta = \sum_{m=1}^Z \gamma(T_{\text{gate}}(Q_1 l_m' + f_1)) \odot T(Q_1 l_m' + f_2) \quad (3)$$

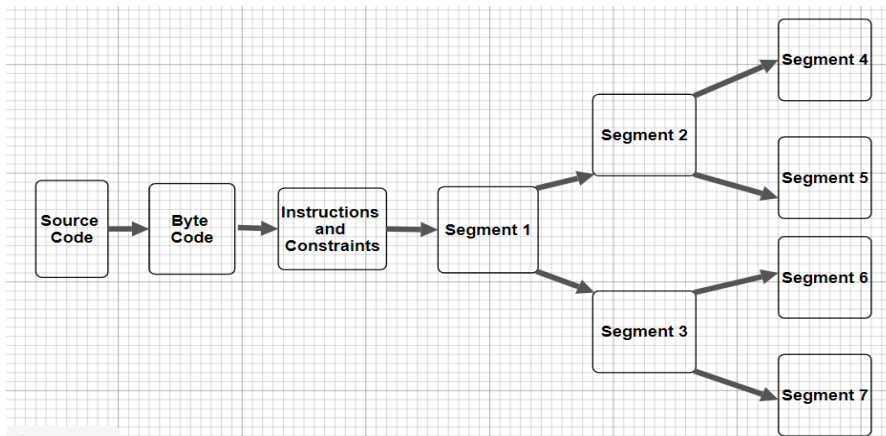


Figure 3. Processing of graph

3.3.1. Framework of model distillation

The framework proposed for model distillation is developed, that focuses on the stage of enhancing smart contract vulnerability identification mutually for byte code. A model distillation framework on the basis of double modality is represented as F that has two input graphs, where $\vartheta_{\text{source}}$ and ϑ_{byte} are inputs. Semantic extraction is developed in this model that utilizes a pooling method for analysis of graphical embedding. ReLU function, Normalization as well as pooling are implemented after every layer that focuses on the required elements to doge overfitting. The embedding of the graph is directed to result in a semantic depiction that is intermediate to the byte code as well as the source code, l_x^f and l_x^w that is combined and known as $l_x = l_x^f \oplus l_x^w$. This characteristic vector is combined for l_x fed in a completely linked layer via the activation function sigmoid using a named output as E_x .

3.3.2. Model distillation network using one modality

The given network denoted W has embeddings as ϑ_{byte} input. The model distillation has a sub network with a particular modification to enhance the information transmission cross modality. The depictions are studied by the model distillation network, for transmission of a model inside the byte code of the model distillation framework. The modality shows an error loss by φ where the modality of byte code is shown as given (4).

$$\mu_\varphi = \sum_{m=1}^r ||l_x^f(f_m) - l_w^f(f_m)||^2 \quad (4)$$

For the (4), the functions are represented by R , the loss is shown as φ for modality of byte code, to broaden the link of cross modality between the byte code as well as the source code. An average method of pooling is used to show the input embedded graph. The idea of source code l_x^w in F as byte code for l_w^f is taken into consideration. The major focus of this condition is the depiction of global text in similar modalities that are paired to the other. A transfer window is used as a layer for W as the capacity of re-developing the

intermediate depiction for features of byte as well as source code. The loss of transfer in a model is built to cross F and W which are two modalities.

$$\mu_{\beta} = \sum_{m=1}^R ||l_x^w(f_m) - l_w^w(f_m)||^2 \quad (5)$$

Considering the similarity in F, $l_w^f(f_m)$ and $l_w^w(f_m)$ are integrated in W, the feature l_w that is integrated is linked to the activation function sigmoid to the output E_w .

3.3.3. Cross-modality model distillation data transfer

The data transfer for F and W, a mutual method of learning is used for training. To analyse the loss that occurs during mutual learning while that uses the entropic loss for data distillation networks E_x , E_w comparative to the truth E. The loss that occurs in E and W is shown by μ_o^v and μ_o^u respectively. We combined the losses and gain the loss of two networks by the equations given in (8). The network attributes are given as ϕ , ρ and γ that balance different losses. The proposed framework is focused where W learns from F and vice versa to gain the method of cross modality ensuring there is performance enhancement.

$$\mu_o^v = \text{loss}(E, E_x) + \text{loss}(E_x, E_w) \quad (6)$$

$$\mu_o^u = \text{loss}(E, E_w) + \text{loss}(E_w, E_x) \quad (7)$$

$$\mu_v = \phi_x \mu_o^v + \rho \mu_{\phi} + \gamma_x \beta \quad (8)$$

$$\mu_u = \phi_x \mu_o^u + \rho \mu_{\phi} + \gamma_v \beta \quad (9)$$

3.4. Architecture of proposed system

3.4.1. Sequential function invocation analysis

The technique proposed in this study aims at surveying sequential invocations in a smart contract that uses various functions. At the beginning phase, a detailed information flow analysis is performed to show the interdependencies in the functions. Although, a ranking system based on priority is developed to show the sequence in which every function has to be called. This sequential order for calling functions is generated. To modify the process as well as use higher intricate phases, the proposed system uses a method of extension that efficiently extends the starting sequence. By using this method, a complete examination is also performed of the function call sequences, that help in the assessment systematically as well as testing of complicated contracts. The proposed methods focus on the challenges of prior existing methods to generate the function call sequences for smart contracts. In this proposed study over 12,000 real world smart contracts are analysed, it is noticed that the type of contract depends on global parameters along with the operating functions such as write or read, on these parameters. Choosing random function neglects these links. The functions that implement the write operation are given higher priority because of the impact of changing state, unlink the read operations. Initialization of a parameter has no effect on the process. The technique of prioritization modifies the knowledge of contract phase space as well as its branches, which proposes an efficient method for function call.

3.4.2. Enhanced test case development

The proposed study utilizes a technique to aid the beginning test cases to be mutated, focusing on the resulting generated cases to have a prior defined objective branch. This is aided using a distance-based metric for branch, that measures the closeness of the test case to meet the constraints of the desired objective branch. By iterative advancements, the proposed system systematically fine tunes the test cases, by improvising its alignment with the set of conditions by the objective branch. By using this method, the technique improves the effectiveness of generating the test case that navigates the flow of the program to particular branches, that contribute to more effective system testing. The proposed study uses iterative advancements of test cases for function calling sequences. It is initiated by a null set as well as test cases. This includes two major loops, namely one for function call, addition of test cases that grasp new branches into the set, and the other is for iterative advancements of test cases that use branch measure based on distance. This quantifying chooses test cases that are close to the constraints of the new branch for mutation. Mutation function () is used for the mutation process, where generation of mutated test case are validated. The process prolongs inside a limit pf energy till a set of test cases with high quality is obtained, efficiently examining as well as fine tuning the function call sequences. Table 1 shows the algorithm for choosing test cases.

Test case prioritization: the proposed method uses a selection method at first, the executions of test cases are tracked as well as their branches are traversed. The set consists of test cases while they cover prior

unexamined branches. As efficient in showing various branches, this technique is ineffective for branches with high complexity as well as strict constraints.

Table 1. Algorithm for choosing test cases

Algorithm for choosing Test Cases	
Input	Test case α , Vulnerable statements J , Program A
Step 1	f_v RU Run (A, α)
Step 2	$\tau_v \leftarrow \emptyset$;
Step 3	$\tau_x \leftarrow \emptyset$;
Step 4	RQ ;
Step 5	MQ ;
Step 6	While m is lesser than $ f_v $ do If $IsCondIns(m, g_r)$ then $RR + 1, f_r \leftarrow f_v[0 \dots m + 1]$; $g, phase \leftarrow SInfer(f_{prior})$; if $VSR(A, U, g, \aleph)$ then $\tau_x.Add(f_v)$;
	$m \leftarrow m + 1$;
Step 7	If R is greater than 2 then $\tau_x.Add(f_v)$
Step 8	Output: τ_v and τ_x

3.5. Allocation of computational resources

Considering the proposed methodology, the technique is also aimed at lesser frequent as well as possibly vulnerable branches of program. This is a result of an algorithm of branch search being integrated, that calculates the branches that are trained as well as detects the crucial ones that include t and possible x branches. However, the proposed method introduces a technique to allocate energy, using an energy schedule that is customized. This method of assigning is aided by two constraints that are adaptable and are efficiently channelled for allocating the energy to the branches. These elements are utilized, which improves the capacity of the methodology to detect as well as completely test the crucial branches, specifically the ones that are not very common or prone to vulnerabilities, this leads to a higher understandable process of testing.

4. PERFORMANCE EVALUATION

The ESC is utilized for conducting a performance analysis. This analysis involves comparing the proposed mechanism with existing methods and evaluating various vulnerabilities, including code injection, timestamp dependence, and reentrancy. The outcomes are presented in the form of tables and graphs.

4.1. Dataset details and comparison methods

The dataset known as ESC comprises a total of 307,396 functions. The following functions were derived from a dataset consisting of 40,932 smart contracts. There are a total of 5,013 functions that include at least one call statement. The inclusion of value within the system may make it vulnerable to reentrant vulnerabilities. Furthermore, the BLOCK feature is also accessible. The system comprises a grand total of 4,833 routines that incorporate TIMESTAMP instructions. This indicates the possibility of encountering issues associated with timestamp dependencies. Moreover, the DELEGATECALL command is employed in a grand total of 6,896 routines, indicating the possible existence of code injection vulnerabilities. The comparison methodologies used are A two-layer recurrent neural network (RNN) called Vanilla-RNN [26] uses recursion to update its hidden state after receiving a series of codes as input. When code sequences are constantly read, the widely used RNN LSTM [27] regularly modifies the unit state. Gated recurrent unit (GRU) [28]: a gated loop unit that processes code sequences using a gating mechanism. Using a Laplacian layered convolution, graph convolutional networks (GCN) [29] conducts a layered convolution of the input graph. The dataset known as ESC [30] comprises a total of 307,396 functions.

4.2. Re-entrancy

The re-entrancy vulnerability is evaluated with performance metrics of various methods as measured by Accuracy, Precision, Recall, and F1 score. Starting from Vanilla-RNN, which shows an accuracy of 49.64% and an F1 score of 50.71%, which depict lowest performance. LSTM and GRU show incremental improvements in all metrics, with GRU achieving a better recall value of 71.3% and F1 score of 60.87%, indicating its stronger capability in identifying true positives compared to Vanilla-RNN and LSTM. A significant rise in performance is observed with GCN, which achieves an accuracy of 77.85% and an F1 score

of 74.15%. This suggests a much better balance between precision and recall, and an overall predictive performance. Evolutionary strategies (ES) show further improvement, with a high accuracy of 89.74% and an F1 score of 85.76%. persistent scatterer (PS) stands out with the highest values in all categories, with an accuracy of 96.57% and an F1 score of 97.87%, suggesting it is highly effective in precision, recall, and overall classification tasks. Upon conclusion PS performs better in comparison with the state-of-art techniques. Table 2 shows the re-entrancy metric evaluation. Figure 4 shows the re-entrancy vulnerability evaluation for different metrics.

Table 2. Re-entrancy metric evaluation

Methods	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
Vanilla-RNN [26]	49.64	49.82	58.78	50.71
LSTM [27]	53.68	51.65	67.82	58.64
GRU [28]	54.54	53.1	71.3	60.87
GCN [29]	77.85	70.02	78.79	74.15
ES [31]	89.74	85.35	86.19	85.76
PS	96.57	93.468	95.67	97.87

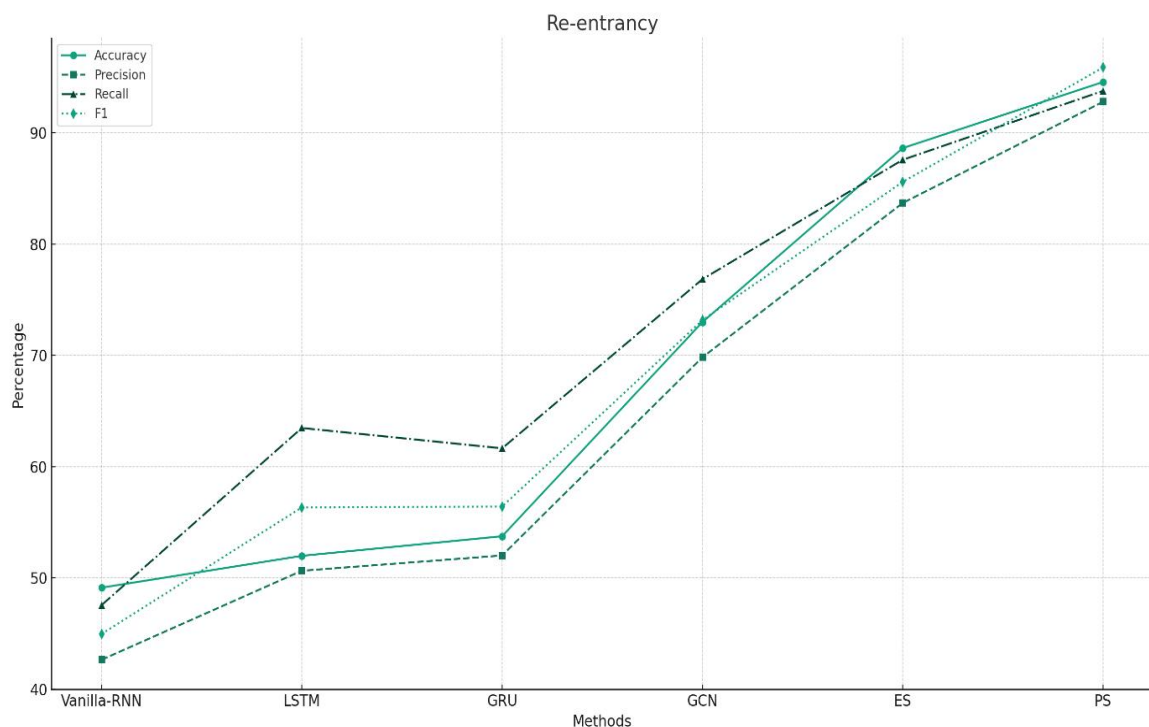


Figure 4. Re-entrancy vulnerability evaluation for different metrics

4.3. Timestamp dependency

The data shows a progression in the performance of various computational methods for timestamp dependency for different metrics. Vanilla-RNN, shows an accuracy of 49.77% and an F1 score of 45.62%, which shows the least effective performance. In comparison, the PS shows an accuracy of 95.76% and an F1 score of 93.46%, indicating a balance of precision and recall, and an overall highly effective classification capability. The LSTM and GRU models show average improvements over Vanilla-RNN, with LSTM achieving a 50.79% accuracy and GRU a slightly better recall of 59.91%. However, both LSTM and GRU have F1 scores showcase average performance for precision and recall. The GCN model represents a significant leap, with an accuracy of 74.21% and a recall of 75.97%, which is indicative of its strong predictive performance, particularly in identifying true positives. The ES method also demonstrates high efficiency with an 88.52% accuracy and a substantial F1 score of 84.1%, signifying a very effective classification performance. Upon conclusion PS performs better in comparison with the state-of-art techniques. Table 3 shows the timestamp dependency metric evaluation. Figure 5 shows the timestamp dependency metric evaluation for different metrics.

Table 3. Timestamp dependency metric evaluation

Methods	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
Vanilla-RNN [26]	49.77	51.91	44.59	45.62
LSTM [27]	50.79	50.32	59.23	54.41
GRU [28]	52.06	49.41	59.91	54.15
GCN [29]	74.21	68.35	75.97	71.96
ES [31]	88.52	82.07	86.23	84.1
PS	95.76	92.37	94.76	93.46

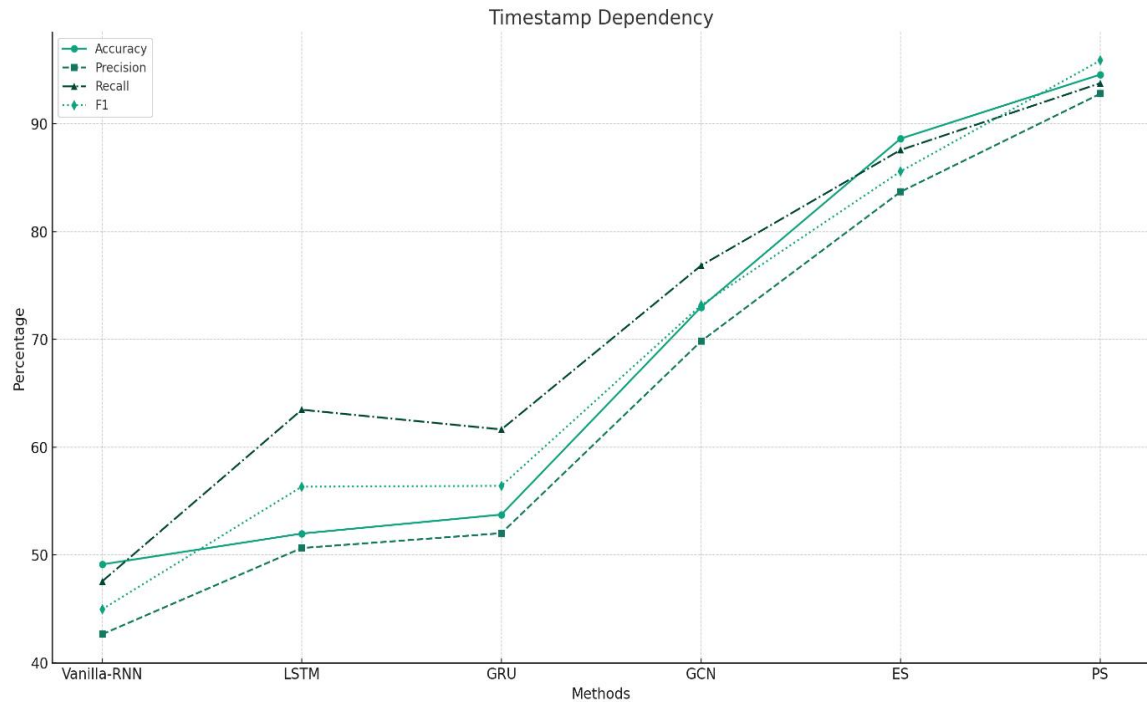


Figure 5. Timestamp dependency metric evaluation for different metrics

4.4. Code injection

Analyzing the given metrics for the listed methods for code injection vulnerability, a progression in performance is observed from Vanilla-RNN to PS. Vanilla-RNN has the lowest scores with an accuracy of 49.12% and an F1 score of 44.96%, a less effective method for both positive identification and balance between precision and recall. LSTM shows an improvement in accuracy to 51.98% and a significant increase in Recall to 63.47%, indicating better identification of true positives. GRU slightly enhances accuracy to 53.74% and maintains a similar Recall to LSTM but improves the F1 score to 56.41%, suggesting a better balance between precision and recall compared to both Vanilla-RNN and LSTM. GCN takes a considerable leap, with accuracy increasing to 72.98% and an F1 score of 73.16%. This suggests a stronger overall predictive performance and a better balance between precision and recall. ES demonstrates a significant improvement, achieving an accuracy of 88.62% and an F1 score of 85.58%, showing it to be a highly effective classification method. Finally, PS stands out with the highest accuracy at 94.56% and an exceptional F1 score of 95.87%, indicating its higher performance across all metrics. Table 4 shows the code injection metric evaluation. Figure 6 shows the code injection metric evaluation for different metrics.

Table 4. Code injection metric evaluation

Methods	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
Vanilla-RNN [26]	49.12	42.64	47.55	44.96
LSTM [27]	51.98	50.64	63.47	56.33
GRU [28]	53.74	52.01	61.64	56.41
GCN [29]	72.98	69.82	76.84	73.16
ES [31]	88.62	83.69	87.57	85.58
PS	94.56	92.78	93.76	95.87

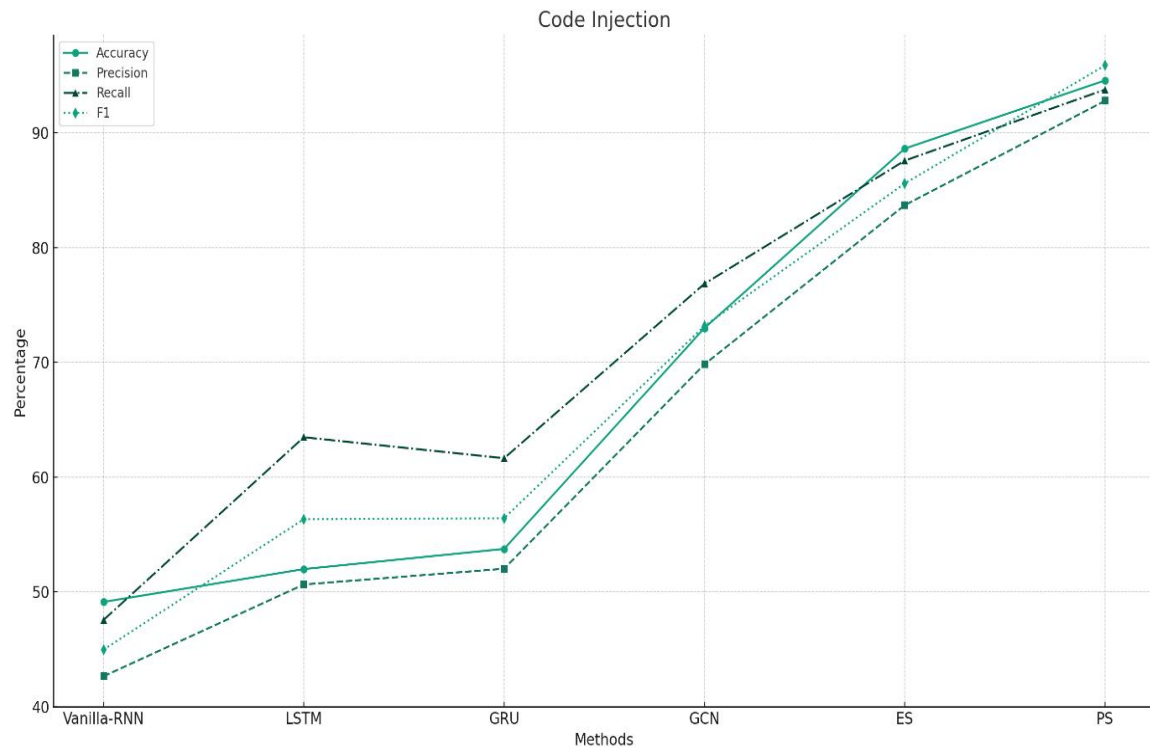


Figure 6. code injection metric evaluation for different metrics

5. CONCLUSION

In conclusion, the proliferation of blockchain technology and the integral significance of smart contracts within its ecosystem have underscored the imperative requirement for robust vulnerability detection mechanism. The proposed methodology, characterized by the GAMDI-Net, marks a significant advancement in smart contract vulnerability detection. The comprehensive framework integrates a graphical learning module, dual-modality model distillation, and a mutual modality learning mechanism to address the complexities of smart contract code and its inherent vulnerabilities. Empirical evaluations, as demonstrated in the performance analysis against the ESC dataset, validate the efficiency of GAMDI-Net in surpassing state-of-art methods. The methodology not only delivers higher accuracy in pinpointing vulnerabilities such as re-entrancy, timestamp dependency, and code injection but also establishes a new benchmark for future research in blockchain security. By fostering a deeper understanding of smart contract vulnerabilities and providing a robust mechanism for their detection, GAMDI-Net contributes to the enhancement of trust and safety in blockchain technology implementations.




REFERENCES

- [1] C. F. Torres, M. Steichen, and R. State, "The art of the scam: Demystifying honeypots in ethereum smart contracts," *Proceedings of the 28th USENIX Security Symposium*, pp. 1591–1607, 2019.
- [2] R. Baldoni, E. Coppa, D. C. D'elia, C. Demetrescu, and I. Finocchi, "A survey of symbolic execution techniques," *ACM Computing Surveys*, vol. 51, no. 3, 2018, doi: 10.1145/3182657.
- [3] Y. Fu *et al.*, "EVMFuzzer: Detect EVM vulnerabilities via fuzz testing," *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1110–1114, 2019, doi: 10.1145/3338906.3341175.
- [4] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, "Eth2Vec: learning contract-wide code representations for vulnerability detection on Ethereum smart contracts," *BSCI 2021 - Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure, co-located with ASIA CCS 2021*, pp. 47–59, 2021, doi: 10.1145/3457337.3457841.
- [5] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "ContractWard: automated vulnerability detection models for Ethereum smart contracts," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1133–1144, 2021, doi: 10.1109/TNSE.2020.2968505.
- [6] S. Badriddoja, R. Dantu, Y. He, K. Upadhayay, and M. Thompson, "Making smart contracts smarter," *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2021*, 2021, doi: 10.1109/ICBC51069.2021.9461148.
- [7] P. Tsankov, A. Dan, D. Drachler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 67–82, 2018, doi: 10.1145/3243734.3243780.





- [8] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, "Evaluating explanation methods for deep learning in security," *Proceedings - 5th IEEE European Symposium on Security and Privacy, Euro S and P 2020*, pp. 158–174, 2020, doi: 10.1109/EuroSP48549.2020.00018.
- [9] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural networks," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2021, pp. 3283–3290, 2020, doi: 10.24963/ijcai.2020/454.
- [10] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," *Proceedings - 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB 2019*, pp. 8–15, 2019, doi: 10.1109/WETSEB.2019.00008.
- [11] J. Gao, H. Liu, C. Liu, Q. Li, Z. Guan, and Z. Chen, "EASYFLOW: Keep ethereum away from overflow," *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion 2019*, pp. 23–26, 2019, doi: 10.1109/ICSE-Companion.2019.00029.
- [12] K. Bhargavan *et al.*, "Formal verification of smart contracts: Short paper," *PLAS 2016 - Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, co-located with CCS 2016*, pp. 91–96, 2016, doi: 10.1145/2993600.2993611.
- [13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 13-17-August-2016, pp. 785–794, 2016, doi: 10.1145/2939672.2939785.
- [14] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997, doi: 10.1006/jcss.1997.1504.
- [15] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.
- [16] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999, doi: 10.1023/A:1018628609742.
- [17] J. Cheng, J. Song, D. Fan, Y. Zhang, and W. Feng, "Current status and prospects of blockchain technology," *Communications in Computer and Information Science*, vol. 1253 CCIS, pp. 674–684, 2020, doi: 10.1007/978-981-15-8086-4_63.
- [18] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus in the lens of blockchain," *arXiv-Computer Science*, pp. 1-23, 2018.
- [19] Y. Ni, C. Zhang, and T. Yin, "A review of approaches for detecting vulnerabilities in smart contracts within web 3.0 applications," *Blockchains*, vol. 1, no. 1, pp. 3–18, 2023, doi: 10.3390/blockchains1010002.
- [20] Y. Zhang *et al.*, "An efficient smart contract vulnerability detector based on semantic contract graphs using approximate graph matching," *IEEE Internet of Things Journal*, vol. 10, no. 24, pp. 21431–21442, 2023, doi: 10.1109/JIOT.2023.3294496.
- [21] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967, doi: 10.1109/TIT.1967.1053964.
- [22] Z. Ying-li, M. Jia-li, L. Zi-ang, L. Xin, and Z. Rui, "Overview of vulnerability detection methods for Ethereum solidity smart contracts," *Computing Sciences*, vol. 49, no. 3, pp. 52–61, 2022.
- [23] F. Ma *et al.*, "Security reinforcement for Ethereum virtual machine," *Information Processing and Management*, vol. 58, no. 4, 2021, doi: 10.1016/j.ipm.2021.102565.
- [24] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H. N. Lee, "Systematic review of security vulnerabilities in Ethereum blockchain smart contract," *IEEE Access*, vol. 10, pp. 6605–6621, 2022, doi: 10.1109/ACCESS.2021.3140091.
- [25] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, and X. Wang, "Combining graph neural networks with expert knowledge for smart contract vulnerability detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 1296–1310, 2023, doi: 10.1109/TKDE.2021.3095196.
- [26] C. Goller and A. Kuechler, "Learning task-dependent distributed representations by backpropagation through structure," *IEEE International Conference on Neural Networks - Conference Proceedings*, vol. 1, pp. 347–352, 1996, doi: 10.1109/icnn.1996.548916.
- [27] T. Zia and U. Zahid, "Long short-term memory recurrent neural network architectures for Urdu acoustic modeling," *International Journal of Speech Technology*, vol. 22, no. 1, pp. 21–30, 2019, doi: 10.1007/s10772-018-09573-7.
- [28] A. A. Ballakur and A. Arya, "Empirical evaluation of gated recurrent neural network architectures in aviation delay prediction," *Proceedings of the 2020 International Conference on Computing, Communication and Security, ICCCS 2020*, 2020, doi: 10.1109/ICCCS49678.2020.9276855.
- [29] S. Fu, W. Liu, D. Tao, Y. Zhou, and L. Nie, "HesGCN: Hessian graph convolutional networks for semi-supervised classification," *Information Sciences*, vol. 514, pp. 484–498, 2020, doi: 10.1016/j.ins.2019.11.019.
- [30] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of ethereum smart contracts," *Proceedings - International Conference on Software Engineering*, pp. 9–16, 2018, doi: 10.1145/3194113.3194115.
- [31] Z. Liu, M. Jiang, S. Zhang, J. Zhang, and Y. Liu, "A smart contract vulnerability detection mechanism based on deep learning and expert rules," *IEEE Access*, vol. 11, pp. 77990–77999, 2023, doi: 10.1109/ACCESS.2023.3298048.

BIOGRAPHIES OF AUTHOR







Dr. Preethi    received a bachelor's degree in computer science and engineering from VTU, Karnataka in 2008, a master's degree in computer science and engineering from VTU, Karnataka 2013, and a philosophy of doctorate degree in Computer Science and Engineering from Presidency University, Bangalore in 2022, respectively. She has a total of 15 years of Teaching experience. She is currently working as an Assistant Professor-Senior Scale in the Department of Information Technology, Manipal Institute of Technology, Bengaluru, Manipal Academy of Higher Education, Manipal, India. Her research areas include the internet of things, computer architecture, and cryptography. She has many papers to her credit in reputed international journals, national journals, and conferences. She has been serving as a reviewer for highly respected journals. She can be contacted at email: preethi.srivaths@manipal.edu.







Dr. Mohammed Mujeer Ulla     currently working as an Assistant Professor-Senior Selection Grade in the School of Computer Science and Engineering since 2017. He is an alumnus of R.V College of Engineering- Bangalore in his UG and PG. He received the philosophy of doctorate degree in Computer Science and Engineering from Presidency University, Bangalore, respectively. He has many papers to his credit in reputable international journals, national journals, and conferences. He has been serving as a reviewer for highly respected journals. His areas of expertise include the internet of things and wireless sensor networks. He can be contacted at this email: mujerroshan@gmail.com.







Dr. Ashwitha Anni     holds a Bachelor of Engineering (B.E.) in Information Science and Engineering, Master of Engineering (M.E) in Software Engineering and Ph.D. in Computer Science and Engineering on Machine Learning, besides several professional certificates and skills. She is currently working as an Assistant Professor (Senior) at Information Technology Department in Manipal Institute of Technology (MAHE), Bangalore, Karnataka, India. Her research areas of interest include data science, artificial intelligence, machine learning, and deep learning. She has published multiple papers in international journals and conferences, from June 2016 to November 2023. She can be contacted at email: ashwitha.a@manipal.edu.



Pavithra Narasimha Murthy     received her M.Tech. degree in Computer Science and Engineering from Dayanand Sagar College of Engineering (VTU) and B.E. degree in Information Science and Engineering from VTU. She is currently working as an Assistant Professor, Department of Computer Science and Engineering, Manipal Institute of Technology, Bengaluru, Manipal Academy of Higher Education, Manipal India. Her research interests are computer vision and pattern recognition, generative AI, data mining, and big data analytics. She has around 14.6 years of teaching experience. She can be contacted at email: pavithra.apr02@gmail.com.



Sapna Renukaradhya     is currently working as Assistant Professor in Manipal Institute of Technology, Bengaluru. She obtained her B.E. from SJBIT and M.Tech. from Dr. AIT and is pursuing her Ph.D. under VTU. Her areas of interest include semantic web, machine learning, IoT, and data mining. She can be contacted at email: sapna.r@manipal.edu.