Babylonian Journal of Machine Learning
Vol.2026, **pp**. 64–96
DOI: https://doi.org/10.58496/BJML/2026/009; ISSN: 3006-5429
https://mesopotamian.press/journals/index.php/BJML

Research Article

HGS-RF: A Heuristic-Guided Selective Random Forest for Multi-Label Vulnerability Detection in Smart Contracts

Saad Nasser AlAzzam [1,*], Ghassan ALDharhani [1], Raenu ALKolandaisamy [1]

[1] *UCSI University ICDS, 65000, Cheras 56000, Malaysia.*

ARTICLE INFO

Article History

Received 19 Jan 2026
Revised 22 Feb 2026
Accepted 30 Mar 2026
Published 20 Apr 2026

Keywords
Artificial Intelligence,
Smart contracts,
Roberta,
Vulnerability detection,
Security analysis,
Sustainable Digital Infrastructure,
Sustainable Growth.

ABSTRACT

Smart contracts execute the transactions in the blockchain network. Smart contracts are prone to various types of security vulnerabilities, which may lead to huge financial losses. The detection of smart contract vulnerabilities is mainly focused on binary classification, where smart contracts are classified as either vulnerable or non-vulnerable. The detection of smart contract vulnerabilities is also focused on multi-class classification, where each smart contract is classified into different types of vulnerabilities. Smart contracts in the real world have various types of vulnerabilities; hence, multi-label detection of smart contract vulnerabilities is required. There are a limited number of studies that have addressed the topic of multi-label classification. In order to overcome the above-mentioned limitation, a new hybrid framework, namely Heuristic Guided Selective Random Forest (HGS-RF), for the detection of multi-label smart contract vulnerabilities has been proposed in the present research work. To achieve the above-mentioned objective, the present research proposes a new framework that integrates the transformer-based feature extraction method with the RoBERTA architecture and the multi-stage-based method for the detection of smart contract vulnerabilities. The framework that is being proposed utilizes a few-shot learning module based on 'Prototypical Networks' for binary vulnerability detection. This utilizes a Multilayer Perceptron (MLP) to map the contract embeddings into a metric space, where the 'vulnerable' and 'healthy' class prototypes are computed based on the support samples, and the contracts are classified based on their Euclidean distance from these computed prototypes, with final predictions refined using k-nearest neighbors (KNN). Second, the proposed HGS-RF architecture utilizes an inferential filtering mechanism that mimics the principle of passive selection of the human immune system. This inferential filtering mechanism can be considered a layer that filters out weak decision trees generated during the initial detection phase. Thus, the inferential filtering mechanism filters out weak decision trees and retains high-performing trees capable of accurately identifying malicious samples from smart contracts. This selective filtering process enables the model to focus on informative vulnerability patterns, thereby improving classification reliability and reducing false positives in multi-label vulnerability detection. Experimental evaluation of the proposed framework, with up to 37 types of vulnerability included, proves its efficiency. F1 scores of up to 0.99 were obtained for the binary classification stage, and the HGS-RF model achieved an accuracy of up to 0.97 in detecting multi-label vulnerability . These results demonstrate that the proposed approach improves vulnerability detection accuracy while enabling more comprehensive security analysis of smart contracts.

1. INTRODUCTION

Smart contracts are self-executing programs deployed on blockchain platforms such as Ethereum, where they automate and enforce financial transactions between parties [1]. Once deployed, smart contracts cannot be modified, making their initial security posture critical [2] [3] [4]. The increasing adoption of smart contracts for high-value transactions has made them attractive targets for attackers, yet vulnerabilities frequently arise due to the complexity of smart contract languages like Solidity and the absence of standardized secure development practices [5] [6] [7]. Undetected vulnerabilities can lead to catastrophic financial losses and erode trust in blockchain systems. Traditional security evaluation methods, including manual code reviews and conventional analysis tools, are time-consuming, labor-intensive, and often fail to detect complex or context-dependent vulnerabilities [8]. The existing methods for the detection of vulnerabilities in smart contracts are either based on a binary classification paradigm, in which a smart contract is classified as vulnerable or non-vulnerable, or a multi-class classification paradigm, in which a single type

*Corresponding author. Email: 1002372379@ucsiuniversity.edu.my

of vulnerability is predicted for a smart contract. The existing methods have significantly contributed to the development and advancement of the field of smart contract security. However, the existing methods are not sufficient to handle the complex reality of smart contract security. In the complex reality of smart contract security, a single smart contract may contain multiple types of vulnerabilities. For instance, a single smart contract may contain both re-entrancy and integer overflow types of vulnerabilities. However, the existing methods are not capable of detecting multiple types of vulnerabilities in a single smart contract. The existing methods are either not capable of detecting different types of vulnerabilities, as in the binary classification paradigm, or are capable of detecting a single type of vulnerability, as in the multi-class classification paradigm. A new method for the detection of vulnerabilities in smart contracts, which is capable of detecting multiple types of vulnerabilities in a single smart contract, is proposed in this paper. The proposed methodology in this research is based on building a deep learning model using the Python programming language to detect vulnerabilities in smart contracts. Transfer learning is used, especially taking advantage of prior knowledge and semantic representations of words learned in the Roberta models [9]. This model is used to extract features from smart contracts. For binary classification, the few-shot learning method has been employed with the 'Prototypical Networks' method. Once the features have been extracted by employing the 'RoBERTa' method, the 'Multilayer Perceptron' is used for mapping the features into a lower-dimensional metric space. The prototypes for 'vulnerable' and 'healthy' class contracts will be the mean embeddings of 'support' set contracts for each class. The class label for the 'query' contract will be the class that is closest in Euclidean distance to the prototypes for that class. This method is efficient for 'vulnerable' contract detection. If the contract is infected with a vulnerability, it is entered into a multi-class classification process to determine the type of vulnerability. It is important to note that the framework is best used in the auditing process before the deployment, as opposed to the monitoring aspect. The computational requirement for the feature extraction process, which involves the use of the transformer, is justified in the given scenario, as it is beneficial in the analysis before the deployment of the contract, which would save costly exploits. However, in the event that the application requires the analysis in real time, the proposed framework is flexible and can accommodate alternative approaches in the feature extraction component.

In summary, this paper makes the following key contributions:

1) A novel hybrid approach for multi-label vulnerability detection: We propose a novel hybrid approach called HGS-RF (Heuristic Guided Selective Random Forest), which is an approach that takes into consideration the multi-label problem of multiple existing vulnerabilities within a single smart contract, which has not been adequately addressed in any of the literature relating to binary and multiclass classification problems.
2) Integration of Few-Shot Learning with Prototypical Networks for Binary Vulnerability Detection: In this study, a few-shot learning module was developed by integrating a prototypical network with a multilayer perceptron (MLP) for efficient binary classification and solving problems of class imbalance in detecting emerging patterns of vulnerabilities.
3) A new heuristic filter method using the human immunology system: We propose a new inferential filter mechanism that favors the retention of high-performing detectors in the form of decision trees and discards the poor detectors, making the classifier more reliable and reducing the false alarms in the multi-label detectors.
4) We test our proposed framework on five datasets, including two datasets for multi-label classification, where up to 37 types of vulnerabilities are present. This shows the efficacy of our proposed method for binary, multi-class, and multi-label classification tasks.

2. BLOCKCHAIN

BC is a decentralized system that offers many benefits, such as decentralization, resistance to tampering, and the ability to trace [2]. This technology is suitable for critical applications such as data storage for anti-counterfeiting and data security [10]. In BC, a node is a computing device that joins a P2P network [11] and contributes computing power. All of these nodes depend on the P2P protocol. The network organizes the activities of each node in a decentralized and distributed manner. figure 1 shows the blockchain layer structure [12].

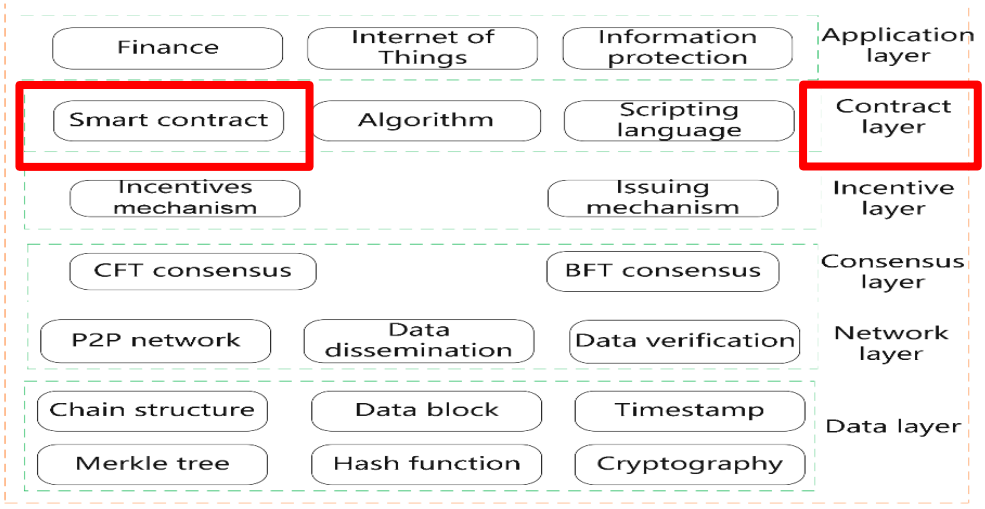

Fig. 1. The blockchain layer structure

The data layer includes the chain structure, data blocks, timestamps, Merkle trees, hash functions, and cryptographic methods. This layer is used for many operations, such as management, organization, and storage. The network layer uses a P2P to connect all nodes to the chain. A consensus layer uses a specific consensus algorithm for determining the nodes that have the right to execute and verify transactions. The incentive layer determines the rewards that are given to the nodes after the correct execution of transactions and the penalties imposed on the nodes that make mistakes. The smart contract layer contains the set of contracts that are being executed. The application layer allows BC to apply the SC code to various real-world scenarios.

2.1 Smart Contracts vs Traditional Contracts

Traditional contracts are agreements between two parties. Usually, these include contract terms relating to an exchange of value. In many cases, they represent a legally binding agreement.

Smart contracts are a type of self-executing contracts that run on a blockchain system like Ethereum. Smart contracts execute their conditions or rules automatically without the need to be controlled. Smart contracts execute in a deterministic way across the entire blockchain network. This ensures the transparency of the executed code. Smart contracts cannot be patched in case of a vulnerability exploit. This is the major security problem that arises due to the immutability of the smart contracts. Smart contracts handle digital assets or financial conditions. This implies that any small mistake in the code can cause a significant security threat.

Common smart contract vulnerabilities include reentrancy attacks, integer overflows, timestamp manipulation, and external calls. Smart contract vulnerabilities arise from the interactions between the smart contracts. Smart contracts operate in a trustless environment. This allows attackers to exploit the vulnerabilities in the smart contracts. Automated vulnerability detection techniques have been found to play a crucial role in the identification of vulnerabilities in smart contracts. Machine learning-based techniques have been found to provide promising solutions for vulnerability detection in smart contracts. This is due to the ability of the techniques to learn from the code representation of the smart contracts.

Table 1 shows a comparison between smart contracts and traditional contracts.

TABLE I. A COMPARISON BETWEEN SMART CONTRACTS AND TRADITIONAL CONTRACTS

Term	Smart Contracts	Traditional Contracts
Definition	Self-executing contracts with terms written in code on a blockchain	Traditional contracts are written in natural language, requiring manual enforcement
Platform	Blockchain platforms (e.g., Ethereum, Binance Smart Chain)	Physical or digital documents (e.g., paper, PDFs, Word files)
Execution	Automatic execution	Requires human involvement for execution and enforcement.
Verification	Verified by blockchain nodes, ensuring transparency and immutability	Verified by legal professionals or parties involved

Cost	Lower cost after deployment (no intermediaries like lawyers or banks)	Higher costs due to legal fees, intermediaries, and administrative overhead
Speed	Near-instantaneous execution	Time-consuming due to manual review and execution
Trust Model	Trustless (relies on code and blockchain)	Based on mutual trust and legal enforceability
Security	Secure but vulnerable to coding bugs and vulnerabilities	Secure but vulnerable to human errors and fraud
Flexibility	Poor flexibility, as it cannot be modified after deployment. So, Smart contracts with vulnerabilities cannot be fixed once deployed.	More flexible as it can be renegotiated or amended.
Accessibility	Requires technical knowledge to develop and deploy	Requires legal knowledge to draft and interpret
Intermediaries	Eliminates intermediaries through automation	Relies heavily on intermediaries like lawyers and notaries.
Examples	Token issuance, decentralized finance (DeFi), and supply chain tracking	Real estate contracts, employment agreements, and service contracts

3. LITERATURE REVIEWS

The identification of vulnerabilities using smart contracts has followed different paradigms, and each of the paradigms has its relative merits and demerits. A synthesized criticism of the existing techniques based on relative methodologies followed, and the research gap that this study aims to fill, are presented in this section.

3.1 Rule-Based Static Analysis Tools

The first approach to ensure the security of smart contracts was static analysis, which is a technique to analyze the contract's code without executing it. The tools for static analysis of smart contracts create guidelines based on specific vulnerabilities. This paradigm is also adopted by Ethainter [13] , which makes use of data log rules for the tracking of information flow in multiple transactions and the detection of complex vulnerabilities. However, the tool does not make use of the detection of new threats, which have not been programmed as a pattern. NeuCheck [14] , on the other hand, converts the source code into a syntax tree and makes use of DOM4j queries for the detection of security patterns. However, the static nature of the tool does not allow the detection of new threats due to the interaction of the contracts. SESCon [15] and SmartScan [16] are based on the same paradigm, with the latter being specifically used for the detection of Denial of Service (DoS) vulnerabilities. One basic limitation is common to all rule-based static analysis tools, namely that the ability of the tools to detect vulnerabilities is limited only by the rule sets to which the tools have been exposed. It is also important to note that the tools cannot generalize new kinds of vulnerabilities or detect context-dependent vulnerabilities.

3.2 Formal Verification and Hybrid Systems

Formal method relies on mathematical proof for the guarantee of the correctness of the contract, but there is a high computational cost for the same. FSPVM-E [17] is a method that combines symbolic execution, theorem proving, and static analysis in a complete integrated framework for formal specifications and interpreters. A completely different hybridization strategy is used in SmartBugs [18] , which integrates 10 existing tools for a wide range of evaluation benchmarks. SmartGraph [19] takes a more visualization-based approach, using UML diagrams to clarify the interaction between functions and contracts, thus facilitating manual debugging, not automated detection. Formal and hybrid systems either demand extensive manual effort (specification writing, diagram interpretation) or aggregate existing tools without fundamentally advancing detection capabilities.

3.3 Dynamic Analysis Tools

The dynamic analysis tools execute the contracts and trace the execution for detecting the vulnerabilities that are encountered during the execution time. ModCon [20] requires the development of state machine specifications for the contracts. This reduces the scalability of the approach. Etherolic [21] is based on the bytecode of the EVM and has an obfuscation-based engine, including contamination tracking and attack indicators. sFuzz [22] has an adaptive fuzzing

approach, including mutation strategies, and SODA [23] has a real-time attack detection approach, including transaction data extraction. Dynamic approaches are able to detect runtime vulnerabilities, but they require an instrumented environment, cannot detect vulnerabilities that were not encountered during testing, and have a performance penalty that is not acceptable for deployment.

3.4 Deep Learning/Machine Learning Methods

Recent developments have leveraged machine learning to learn vulnerability patterns directly from data, providing the ability to generalize beyond predefined rules.

3.4.1 Graph and Neural Network Approaches

Graph neural networks (GNNs) represent smart contract code as graphs capturing syntactic and semantic structures. The model in [24] converts contracts into graphs where master nodes represent function calls and secondary nodes represent variables. DA-GNN [25] enhances this with dual attention mechanisms to focus on relevant code structures. However, GNNs are computationally expensive and require significant resources for training and inference. On the other hand, the approach of SCVDIE-Ensemble [10] is to utilize ensemble learning from various models such as CNN, RNN, and DNN. This improves the robustness of the approach. Nevertheless, this might not be applicable in real-time since different models have to be trained. CodeNet [26] uses a set of images of a fixed size to represent the bytecode, which can be analyzed using CNN. This approach might lose data from complex contracts and requires a lot of substantial memory.

3.4.2 Classification and Specialized Models

ContractWard [27] utilizes different classifiers of machine learning to detect six kinds of vulnerabilities, but it can handle these kinds of vulnerabilities only. Eth2Vec [28] utilizes PV-DM and NLP for code representation, but it may face problems with new kinds of vulnerabilities. The research [29] utilize multi-task learning with shared lower layers and CNN for various kinds of vulnerabilities. The research [30] proposed a CNN and Random Forest hybrid approach for bytecode analysis. In this approach, source code analysis has the potential for detailed analysis. However, the problem of dataset imbalance is not addressed. An MLP-ANN approach is proposed by [31] for addressing the problem of dataset imbalance. In this approach, control flow graph and fault injection are used. However, synthetic faults are not representative of actual world scenarios. An approach is proposed by BiGAS in [32] or reentrancy vulnerability analysis only. This makes it less applicable for other types of vulnerabilities. An approach is proposed by HAM in [33] for detecting five types of vulnerabilities using the attention mechanism. However, the problem of dataset imbalance is not addressed.

3.4.3 Large Language Models (LLMs) for Vulnerability Detection

Recently, the development of large language models such as GPT, Code Llama, and code-related models has introduced new possibilities for the detection of vulnerabilities. These models, which have been pre-trained on huge amounts of code and text data, have shown promising results in the interpretation of the semantics of the code and the generation of explanations. Recently, several works have been conducted to explore the utility of LLMs for smart contract security. This work, named "GPTScan," uses LLMs such as "GPT" for static analysis, thus allowing the detection of the characteristics of the vulnerability before they are confirmed by the analysis of the program [34]. Other works have used LLMs to detect vulnerabilities using a few-shot approach [35], where examples of vulnerable and non-vulnerable code are used. The major advantages associated with LLMs include the lack of need for training data, the ability to provide explanations in natural language for the identified vulnerabilities, and the ability to identify new patterns of vulnerability that are not covered by the current rule sets. The ability of LLMs to comprehend the context and intent of the code makes them very desirable when dealing with complex vulnerabilities that involve business logic rather than syntax.

However, these LLM-based techniques also have a number of limitations that are currently limiting their practical usage in the following areas:

a. High Computational Cost: LLMs are known to consume a lot of GPU resources, which makes their practical usage require high computational cost.
b. False Positive Rates: LLMs can also produce a lot of false positives, which need to be checked.
c. Lack of Guarantees: LLMs do not provide any guarantees about the completeness and correctness of their analysis.

d. Prompt Sensitivity: The detection capability also depends on prompt engineering, which makes their results difficult to reproduce and standardize.

This limitation, therefore, implies that although the approach of LLMs is promising for the detection of vulnerabilities, it is better used as a supporting tool for experts or in a hybrid form. The approach this paper suggests could be considered a complementary approach to the LLM approach, considering the efficiency and specificity of the approach with known computational costs and decision-making processes.

3.5 Comparative Analysis and Research Gap

Table 2 provides a comparative summary of existing approaches across key dimensions.

TABLE II. COMPARATIVE ANALYSIS OF EXISTING VULNERABILITY DETECTION APPROACHES

Approach Category	Detection Paradigm	Multi label detection	Handles Co-occurring Types	Generalizes to Novel Patterns
Rule-Based Static	Pattern matching	✓ (if rules exist)	✗	✗
Formal Verification	Mathematical proof	✗	✓	✓
Dynamic Analysis	Runtime monitoring	✗	✓	✗
Graph Neural Networks	Learned structure	✗	✗	✓
CNN/Ensemble	Learned features	✗	✗	✓
NLP/Sequence Models	Learned semantics	✗	✗	✓

Across all reviewed approaches, three critical gaps emerge:

a. Binary/Multi-Class Paradigm Dominance: All the proposed methods tackle the problem of vulnerability detection as a binary problem (vulnerable/non-vulnerable) or multi-class problem (one class of vulnerability per contract). This is not true since there are many types of vulnerabilities for the contracts.
b. None of these studies has formulated smart contract vulnerability detection as a multi-label classification problem, where each contract can be associated with multiple vulnerability labels simultaneously.

A lack of multi-label vulnerability detection frameworks creates a noticeable gap. Real-world smart contracts often have many vulnerabilities, for instance, a smart contract can have 'reentrancy' and 'integer overflow' vulnerabilities. The current methods, limited to binary or multi-class classification, can, at best, detect only one type of vulnerability for a given smart contract, leaving a gap in the security checks.

This paper aims to fill this gap by proposing a novel framework called HGS-RF, which is tailored for multi-label vulnerability detection in smart contracts. This is achieved by creating specialized detectors for each vulnerability type and combining their outputs through a novel heuristic filtering mechanism inspired by immune system negative selection, our approach can identify multiple co-existing vulnerabilities within a single contract while maintaining high detection accuracy.

Table 3 shows a comparison between reviewed studies.

TABLE III. A COMPARISON BETWEEN REVIEWED STUDIES

Model	**Input type**	**Vulnerabilities covered**	**Multi-label support**
Ethainter	Source + execution traces (rule logs)	-	No
NeuCheck	Source code → AST	Access Control Vulnerability, Reentrancy Attacks, Hash Collision, Integer Overflow/Underflow, Dependence on Predictable Variables	No
SESCon	Source/AST (XPath queries)	SWC-100, SWC-107, SWC-113, SWC-120, SWC-122	No
SmartScan	Source + static patterns + dynamic test	DoS via Unexpected Revert	No

FSPVM-E	Formal spec + bytecode	-	No
SmartBugs	Aggregated: static + symbolic tools	DASP10 Vulnerabilities	No
SmartGraph	Source → UML/graphs	-	No
ModCon	State-machine specs + runtime	Business Logic Flaws	No
Etherolic	EVM bytecode + contamination engine	integer overflow/underflow, reentrancy, and short address attacks	No
sFuzz	Bytecode / transaction inputs	Gasless Send, Exception Disorder, Reentrancy Attacks, Timestamp & Block Number Dependency, Dangerous Delegatecall, Integer Overflow/Underflow, Freezing Ether	No
SODA	Transaction stream + runtime traces	Reentrancy Attacks, Unexpected Function Invocation, Short Address Attack, Tx.origin Authentication Exploits, Unchecked External Calls, Missing Transfer Events, Block Number & Timestamp Dependence	No (real-time alerts)
Smart contract GNN	Graph (CFG/AST)	Reentrancy, Timestamp Dependence, Infinite Loops	No
DA-GNN	Graph + dual-attention	-	No
SCVDIE-ENSEMBLE	Mixed (CNN/RNN/DNN)	-	No
CodeNet	Bytecode → image (fixed-size)	-	No
ContractWard	Feature-engineered (code)	Timestamp Dependence, Reentrancy, Overflow/Underflow, Call Stack Depth, Transaction Order Dependence	No
Eth2Vec	PV-DM embeddings (code-level)	SC Bytecode-Level Vulnerabilities	No
Multi-task model	Shared embedding + task heads	Computational Vulnerabilities, Reentrancy, Unknown Address Issues	No (multi-task ≠ multi-label per contract)
CNN+RF hybrid	Bytecode/CNN features + RF	Opcode-Level Vulnerabilities	No
MLP-ANN CFG	CFG features + MLP	Opcode Features, CFG-Based Detection	No
BiGAS	Custom features for reentrancy	Reentrancy Vulnerabilities	No
HAM	Attention-based encoders	5 SC Vulnerabilities	No
GPTScan	LLM + static analysis hints	diverse (depends on prompts)	can flag multiple but not structured multi-label)
LLM few-shot	static analysis	diverse (depends on prompts)	Yes

4. METHODOLOGY

The proposed methodology uses transfer learning, especially taking advantage of prior knowledge and semantic representations of words learned in the Roberta model [36]. This model is used to extract features from smart contracts. In the binary classification step, a few-shot learning method, 'Prototypical Networks,' was used. The learned features using the 'RoBERTa' model were passed through a 'Multilayer Perceptron' layer to get the embeddings in a lower dimension. For each 'episode,' the prototypes of the 'healthy' and 'vulnerable' classes were computed as the average embeddings of the 'support' samples. The query contracts were classified as belonging to the class to which the query had the minimum Euclidean distance. This method helps in the robust binary classification using fewer data points and captures the semantic similarity between the vulnerable contracts. Figure 2 shows the general structure of the proposed methodology.

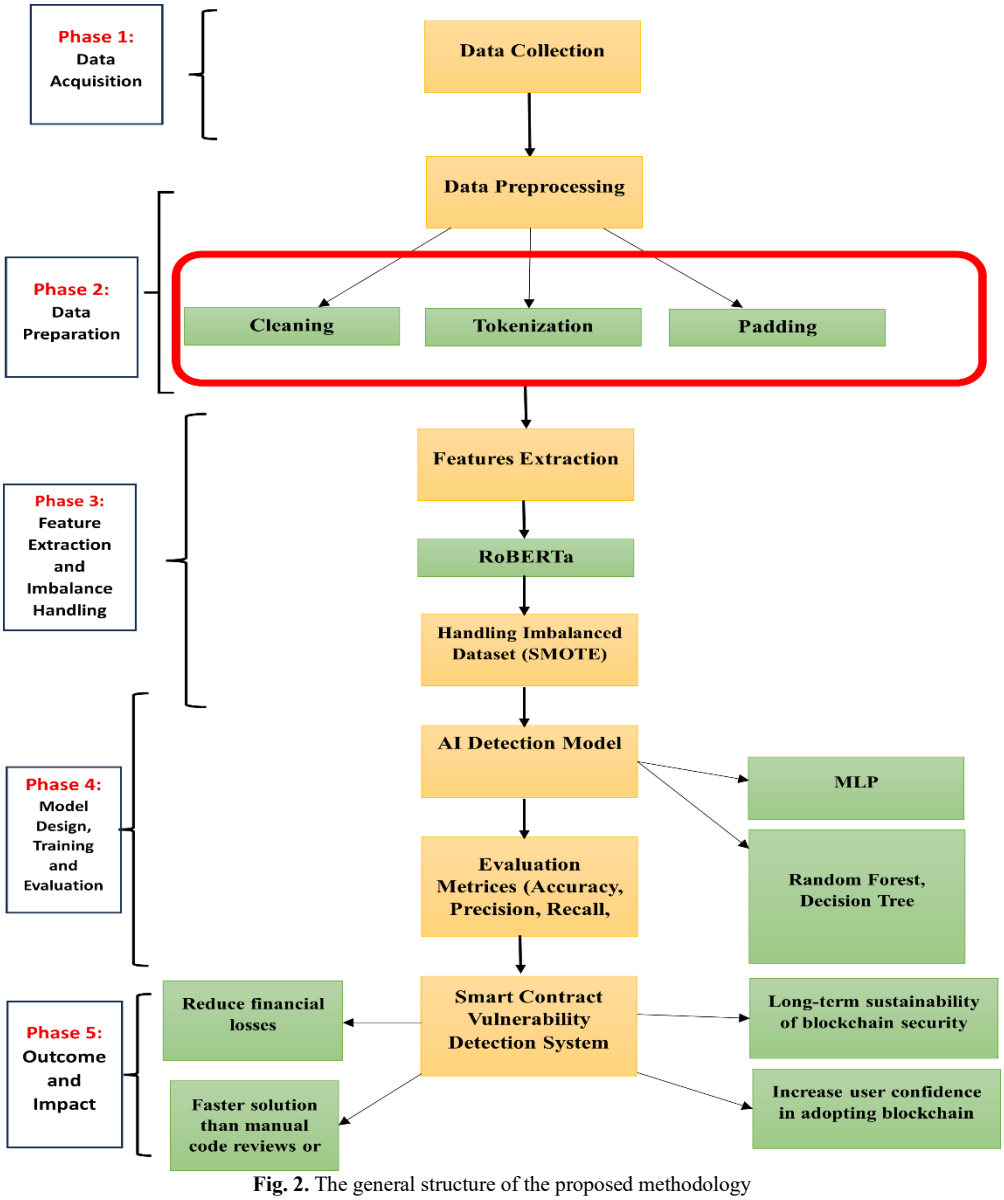

Fig. 2. The general structure of the proposed methodology

4.1 Phase 1: Data collection

In this research, 5 datasets from Kaggle and Huggingface are used. The first dataset contains 36,670 smart contracts (binary classified) into healthy contracts and vulnerable contracts. The second dataset contains 4,285 smart contracts classified into 8 types of vulnerabilities shown in Table 4.

TABLE IV. THE TYPES OF VULNERABILITY IN THE SECOND DATASET

Vulnerability Type	Samples Number
Reentrancy (RE)	1218
Unchecked External Call (UC)	1199
Integer Overflow (OF)	590
Block Number Dependency (BN)	406
Ether Strict Equality (SE)	366
Timestamp Dependency (TP)	312
Dangerous Delegate Call (DE)	97
Ether Frozen (EF)	97

The third dataset contains 35,228 contracts labeled with 9 types of attacks illustrated in Table 5.

TABLE V. THE TYPES OF VULNERABILITY IN THE THIRD DATASET

Vulnerability Type	Samples Number
Reentrancy (RE)	8703
Unchecked External Call (UC)	8539
Safe (SA)	4670
Integer Overflow (OF)	4216
Block Number Dependency (BN)	2894
Ether Strict Equality (SE)	2605
Timestamp Dependency (TP)	2220
Dangerous Delegate Call (DE)	690
Ether Frozen (EF)	691

Datasets 4 and 5 were obtained from Hugging Face and contain multiple types of vulnerabilities. These datasets differ from datasets 2 and 3 in that each smart contract contains several types of vulnerabilities, not just one. In this case, multi-class classification methods fail to capture all vulnerabilities because they only output one type. Many previous research on vulnerabilities in smart contracts has relied on binary and multi-class classification problems, and there is limited work exists have addressed what is called multi-label vulnerabilities, meaning that a single contract contains multiple types of vulnerabilities. Each dataset from 4 and 5 contains 15,000 smart contracts, and each contract contains multiple types of vulnerabilities.

4.2 Phase 2: Data Preparation

4.2.1 Cleaning

The following explains the operations performed in this phase [37]:

- Remove lines that contain comments, punctuation, special characters, and empty lines.
- Convert all letters to lowercase.
- Function and variable names are chosen randomly by the coder. Therefore, these names are meaningless and can cause categorization problems. We converted them to fixed symbolic names such as VAR1 and FUNC1.

4.2.2 Tokenization

In the fields of natural language processing (NLP) and machine learning, the term `Tokenization` refers to the process of converting a string of text into smaller pieces known as tokens [38]. These tokens can be as short as letters or as long as words. The main reason this process is important is that it helps machines understand human language by breaking it down into smaller, more easily parsed chunks. After tokenization, each word is assigned a number [39]. Figure 3 shows the code after tokenization.

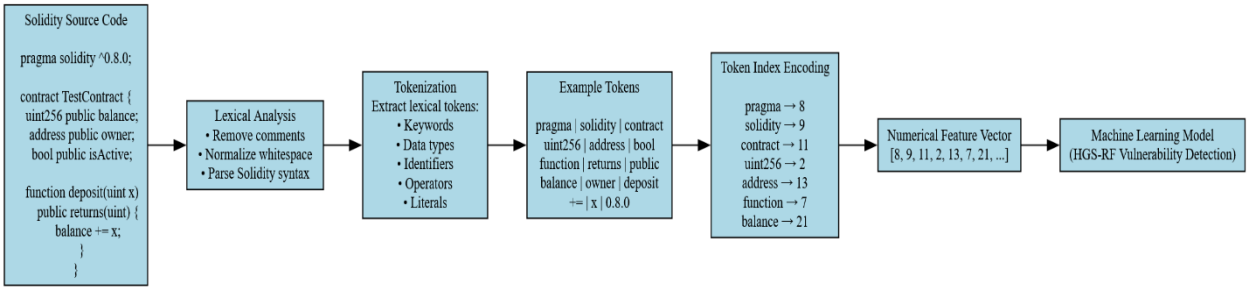

Fig. 3. Tokenization process

4.3 Phase 3: Feature Extraction and Imbalance Handling

At this stage, we have performed feature extraction using the RoBERTa model. Roberta is an excellent model for feature extraction, especially for language processing tasks. Roberta can provide embeddings for each token in a sentence, capturing their meaning in context [40]. The RoBERTa-base transformer model consists of 12 transformer encoder layers, 768 hidden units per layer, and approximately 125 million parameters. The model is pretrained on large-scale text corpora using masked language modeling and is widely used for contextual embedding extraction in natural language processing tasks. In the proposed research, the contextual embedding is obtained from the last hidden layer of the pre-trained model on the RoBERTa-base model. When the smart contract as a sequence of tokens is input to the model, the model provides a contextual embedding vector for each token in the sequence, and the semantic and syntactic relationships between the tokens are represented by the vectors as input feature vectors during the learning stage. The RoBERTa model is used as a frozen feature extractor, meaning that its pretrained weights are not updated during training. Instead, the contextual embeddings generated by RoBERTa are passed to the downstream classifier components.

Each smart contract is tokenized and embedded using the RoBERTa, where, for a smart contract S with token sequence $\{t_1, t_2, \dots, t_n\}$ RoBERTa produces contextual embeddings as shown in formula (1):

$$E = RoBERTa(t_1, t_2, \dots, t_n) \in \mathbb{R}^{n \times d} \quad (1)$$

Where:
$d = 768$: Embedding dimension.
n: number of tokens in that contract.
$\{t_1, t_2, \dots, t_n\}$: A token sequence

In the suggested framework, the RoBERTa model will be employed to generate the contextual embeddings for the tokenized smart contract code. In the suggested framework, for a given smart contract defined by a sequence of tokens $T = \{t_1, t_2, \dots, t_n\}$, the contextualized vector representations can be obtained by the RoBERTa model for the tokens within the sequence of the smart contract's code. Hence, the output of the RoBERTa model for a given smart contract can be defined as a matrix of vector representations $E \in \mathbb{R}^{n \times d}$, where n represents the total number of tokens within the smart contract's code, while d represents the total dimensionality of the contextualized vector representations for the smart contract's code by the RoBERTa model. Additionally, the total length of the smart contracts can vary, hence n can vary depending on the smart contracts.

The use of RoBERTa in feature extraction is a trade-off between the computational cost and the strength of the representation. The computational cost of using the 12-layer transformer model and 768-dimensional embeddings, as in the RoBERTa model, involves 125 million parameters that require a lot of memory during the inference process in the GPU. Although the computational cost of using this model is high, it falls within the limits of what can be considered acceptable during the security analysis before deployment (without internet connection), which is the main

goal of the proposed framework. The semantic strength of contextual embeddings, as in the RoBERTa model, plays a crucial role in increasing the accuracy of the detector, especially in detecting vulnerabilities with complex patterns. In situations where computational resources are limited, the use of Distil-RoBERTa model and static contextual embeddings like Word2Vec and FastText can also be considered, but at the cost of accuracy of detection.

Smart contract vulnerability datasets are typically imbalanced because some vulnerability types occur far less frequently than others. To mitigate this issue, the Synthetic Minority Oversampling Technique (SMOTE) is applied. SMOTE generates synthetic training samples for minority classes by interpolating between existing samples in the feature space [41]. Specifically, for each minority sample, the algorithm identifies its k-nearest neighbors within the same class and generates new synthetic samples along the line segments connecting these neighbors. This approach helps balance the class distribution while preserving the underlying feature structure. SMOTE is applied only to the training portion of the dataset after the train–test split, ensuring that synthetic samples do not influence test data and preventing potential data leakage.

4.4 Phase 4: Model Design, Training, and Evaluation

As mentioned earlier, at this stage, a binary classification process is performed to classify the smart contract into two classes: 0 (healthy contract) and 1 (vulnerable contract). Figure 4 shows the flow chart of the processes at this stage.

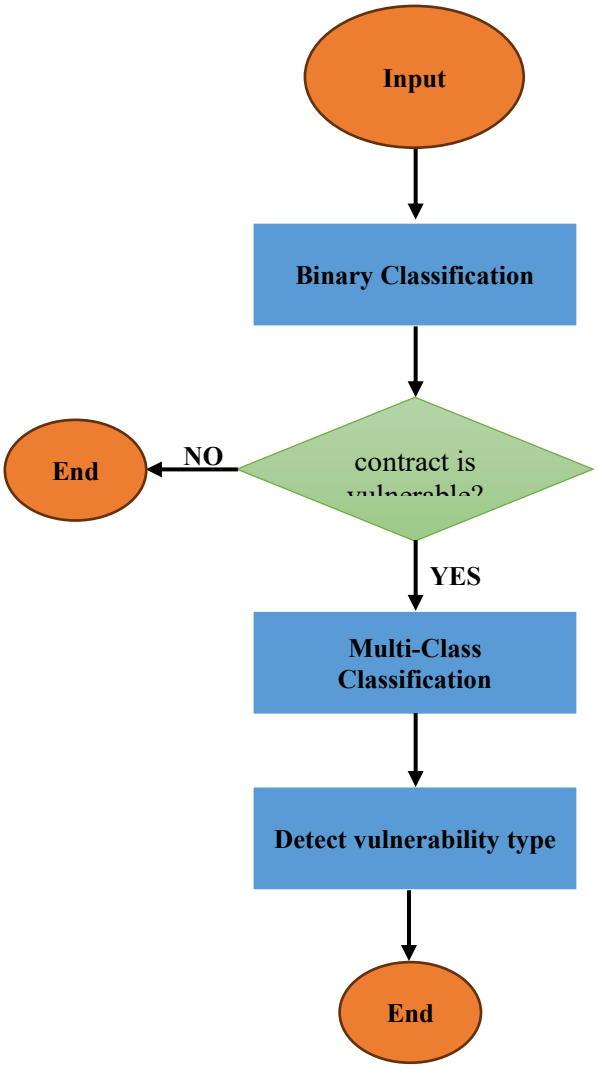

Fig. 4. The flow chart of the processes

4.4.1 Binary classification task

The binary classification step combines the following elements in a cohesive few-shot learning architecture:

- Feature extraction: RoBERTa creates the initial embeddings (dimension 768) for each tokenized contract with n tokens per contract.
- A Multilayer Perceptron (MLP) with three hidden layers (dimension 768 -> 512 -> 128 -> 64) projects the embeddings onto a lower-dimensional space, such that distances are meaningful for the classification task.
- For each episode, the prototypes for the 'healthy' class (class 0) and 'vulnerable' class (class 1) are computed as the mean embedding of the samples from the support set (Formula 3).
- Query samples are classified based on the Euclidean distance to each of the prototypes; the class of the closest prototype determines the predicted class for the query sample (Formula 4).
- The MLP is trained using the cross-entropy loss on the negative distances; the aim is for the embeddings of the same class to cluster around the corresponding prototypes.
- After the MLP has been trained, k-nearest neighbors is applied to the embeddings of the support set for the final classification; this adds a layer of robustness.

In the binary task, the label is defined as shown in formula (2):

$$y \in \{0,1\} : \begin{cases} 0 & Healthy\ contract \\ 1 & Vulnerable\ contract \end{cases} \qquad (2)$$

To perform binary classification, we use few-shot learning over prototypical networks (ProtoNet) [42], which sends the inputs to an embedding space using a Multilayer Perceptron [43] artificial neural network $f_\theta()$ where for each class c, the prototype is the mean embedding of its support set Sc (See formula 3):

$$p_c = \frac{1}{S_c\ count} \sum_{(x_i, y_i) \in S_c} f_\theta(x_i) \qquad (3)$$

Where:
$S_c\ count$: Samples count is support set.
$f_\theta()$: artificial neural network function applied on support samples x_i.

After that, a query sample x_q is assigned to the class whose prototype is closest in Euclidean distance (See formula 4):

$$\hat{y}_q = argmin_c \left(Euclidean - distance\left(f_\theta(x_q), p_c\right)\right) \qquad (4)$$

Where:
$f_\theta(x_q)$: artificial neural network function applied on query samples x_q.
p_c: prototype of the class c.

This formula helps group embeddings of the same class around their prototype, making the robust multi-class classification process easier and faster.
To perform the final classification process, the k-nearest neighbor algorithm is used, which relies on calculating the Euclidean distance of each input sample to its nearest data points and performing the classification based on that.
Figure 5 shows the training inspired by the few-shot learning depending on Prototypes.

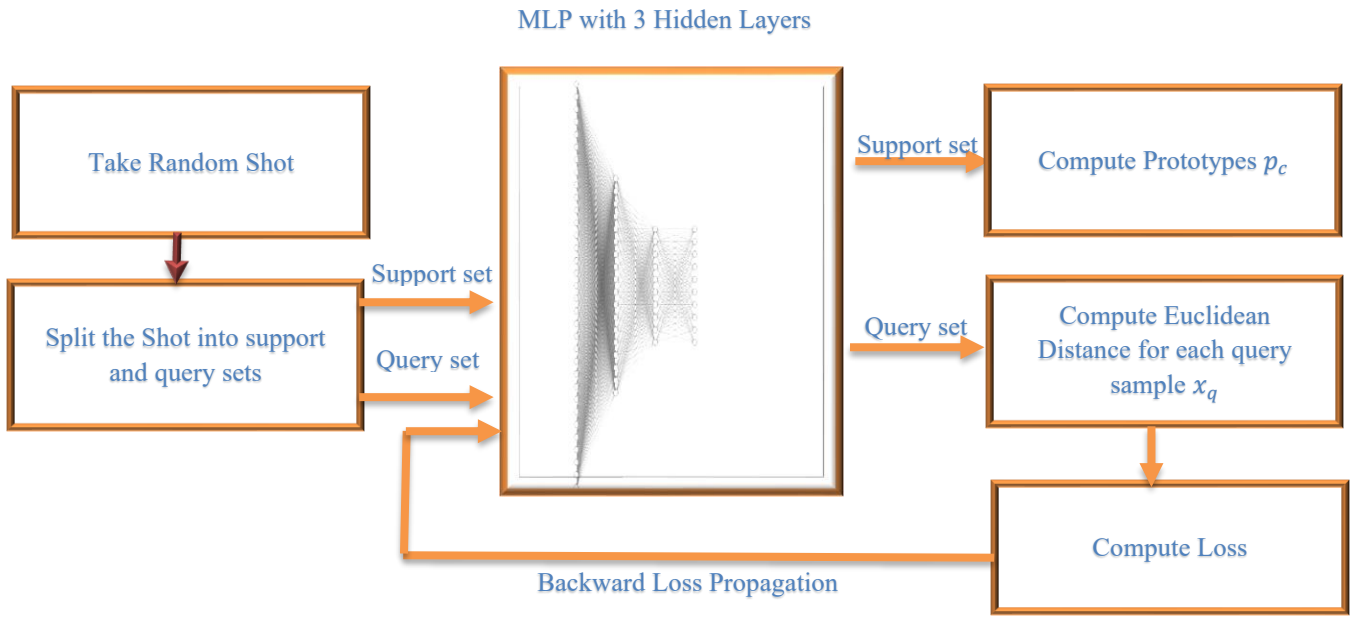

Fig. 5. Training procedure for the binary classification stage using few-shot learning with Prototypical Networks. The MLP learns an embedding space where class prototypes are computed from support samples, and query samples are classified based on distance to these prototypes

Figure 6 shows the model evaluation using few-shot learning with KNN.

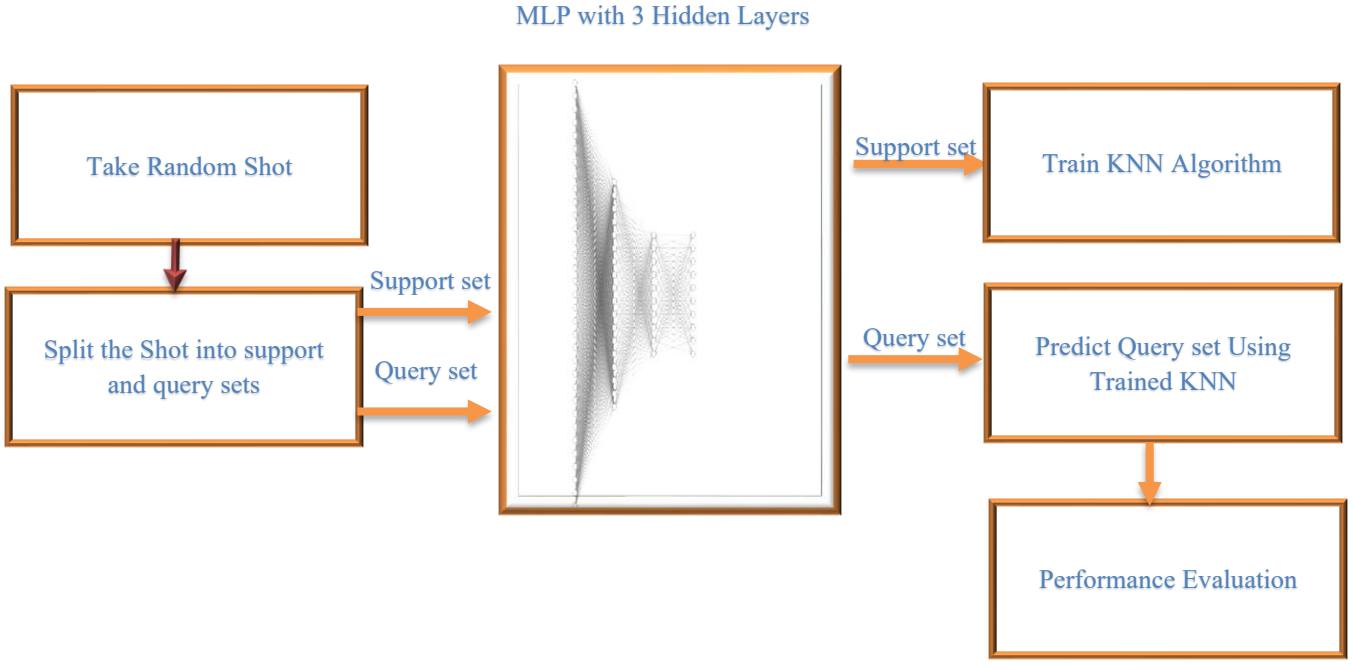

Fig. 6. Evaluation procedure for the binary classification stage. After MLP training, k-nearest neighbors (KNN) is applied to support set embeddings to classify query samples, leveraging the metric space learned during prototype-based training

Figure 5 depicts the training procedure of the proposed model using the Few-Shot Learning (FSL) technique [44]. In this method, a “shot” is first sampled from the dataset, with its size determined by the user so that each class is

equally represented. For instance, if the shot size is set to 5, it includes 5 samples from class 0 and 5 samples from class 1. Subsequently, the shot is randomly divided into two subsets: the Support set and the Query set. The Support set serves as the reference for predicting the labels of the Query set [45].

Both subsets are processed through a Multi-Layer Perceptron (MLP) designed for feature reduction and enhancement [46]. The MLP outputs two transformed groups: support features and query features. This dimensionality reduction preserves essential information while improving computational efficiency, mitigating overfitting, and accelerating model training.

Next, the support features are used to compute class prototypes. For each class, the samples in the Support set are aggregated, and the arithmetic mean of each feature is calculated to generate the prototype representing that class. For example, the prototype for class 0 is obtained by averaging all features of class 0 samples, and similarly for class 1.

Classification is then performed by calculating the Euclidean distance [47] between each query feature and the prototypes. Each query sample is assigned to the class corresponding to the nearest prototype. The predicted labels are compared against the actual labels in the dataset, and the loss function is computed. This loss is backpropagated through the MLP to update its weights accordingly.
This procedure is repeated iteratively: a new shot is sampled from the dataset, and the entire process—from feature extraction to prototype computation and classification—is performed until the specified number of training iterations is completed.
The main goal of the training mechanism shown in Figure 5 is to adjust the MLP network weights by reducing the value of the loss and the representative here with the ranging distance between the Prototype point and the point of the x_q sample by the rear spread of the error.
The proposed MLP includes a set of layers containing artificial neurons. Each neuron in any layer has connections to all neurons in the next layer, which is why it is called dense due to the density of connections between neurons within each layer. Each of these connections has a weight, and the goal of the training process is to adjust these weights to achieve the lowest error in the output. These weights are adjusted at each iteration based on the error between the actual value and the predicted value, represented here by the Euclidean distance between a point representing the sample in a multidimensional space and a reference point representing each class. The proposed MLP consists of an input layer of 768 artificial neurons (matched to the dataset's features without the target feature), followed by a hidden layer of 512 artificial neurons, followed by a second hidden layer of 128 artificial neurons, and an output layer of 64 artificial neurons. The goal of this network is to reduce the feature space from 768 to 64 features before performing classification operations. This is done to improve the representation of features and associate them with weights that can effectively adjust feature values.

In the calculation of loss in Figure 5, the cross-entropy function plays a central role in computing the classification loss. Specifically, the model predicts class labels based on the minimum distance between a xq and a class prototype point. The distances contain the pairwise distances between each xq and the class prototype point, and instead of traditional logits, these distances are passed directly into the cross_entropy function [48]. Although cross-entropy typically expects raw class scores (logits), where higher values indicate greater confidence in a class prediction, this approach leverages the inverse relationship, where the result of Euclidean distance is passed to the cross_entropy function in reverse (-distance), so smaller distances as indicative of higher similarity. Consequently, the function effectively learns to minimize the distance between each input xq and its corresponding class prototype. By computing the cross-entropy loss over distance-based pseudo-logits, the model is guided to bring embeddings of the same class closer together, thereby improving its ability to distinguish between classes. The loss is calculated as shown in formula (5).

$$loss = cross\ entropy\ (-distance, label) \qquad (5)$$

Formula (5) is the cross-entropy loss, but here we're passing distances as logits. Normally, the logits are passed to cross_entropy (higher = more likely), but here, we're passing distances, so lower is better.
So, we invert the distances by passing them directly into the cross_entropy function, and the system assumes higher logits = more likely the lowest distance. So, classes with lower distances (more similar) will result in lower loss, because cross-entropy rewards higher values at the true label index. This makes the model learn to minimize the distance to the correct class prototype point.

Figure 6 depicts the evaluation procedure of the proposed model. Initially, a shot is sampled from the dataset, which is then randomly divided into a Support set and a Query set. Both sets are input into the previously trained MLP model, obtained through the training procedure illustrated in Figure 5. The MLP extracts features from each set, producing Support features and Query features.
Next, the K-Nearest Neighbors (KNN) algorithm [49] is trained on the Support features and subsequently used to predict the labels of the Query features. The predicted labels are then compared with the true labels to assess the model's performance. This process is repeated iteratively: a new shot is drawn from the dataset, and the same steps—from feature extraction to KNN prediction and evaluation—are carried out until the predefined number of iterations is completed.

It is also important to understand the link between the prototype-based classification that takes place during training and the KNN-based classification that takes place during evaluation. As illustrated in Figure 5, during training, the model is trained in an embedding space in which all the samples of a particular class are grouped around their respective prototype due to the ability of the loss function to reduce the distance between the query points and their respective prototypes.
In the evaluation process (see Figure 6), after training the MLP for well-separated embeddings, we apply KNN on the support set for classification. This method makes use of information that is locally available in the embedding space, apart from the global information provided by the prototypes. KNN can be understood as a non-parametric extension that considers multiple neighbors rather than a single prototype, providing robustness to outliers and capturing finer-grained distinctions within classes. These two methods are complementary; one method is used for learning the embedding space, and the other method makes use of the learned embedding space.

4.4.2 Multiclass/ Multilabel classification task

Random Forest constructs an ensemble by generating a large number of decision trees through bootstrap aggregation and random feature selection [50], retaining all trees for the final majority vote. While powerful, this approach is agnostic to the individual quality of each tree, often resulting in a model bloated with weak or redundant learners that consume memory and can degrade overall performance.
The Heuristic-Guided Selective Random Forest (HGS-RF) model proposed in this work introduces a fundamental shift from this inclusive paradigm. The main innovation of this phase is the integration of a heuristic filtering mechanism. This mechanism is inspired by the negative selection of the immune system, which works as a quality gate after generating each tree and before creating the ensemble. This process saves only the most competent decision tree detectors which creates a more efficient ensemble of detectors.
The Human Immune System employs a "negative selection" process where T-cells that react strongly to "self" proteins are eliminated, ensuring that only cells capable of recognizing "non-self" pathogens mature [51]. Inspired by this mechanism, the proposed model treats benign network traffic as "self" and malicious activities as "non-self." Instead of retaining all decision trees like a standard Random Forest, the model introduces a heuristic filter that retains only high-performing tree "detectors" based on their accuracy on "self" (benign) samples. This selective process enhances the ensemble's overall robustness and efficiency.

4.4.2.1 Generation of Candidate Detectors

The model begins by generating a diverse pool of candidate decision tree detectors. To capture nuanced decision boundaries, the multi-class problem is decomposed into multiple binary classification tasks.
For each pair of classes (c_i, c_j), a subset $D_{i,j}$ of the original dataset is generated, containing the samples for each class.

$$D_{i,j} = \{(x, y) \in D \mid y \in \{c_i, c_j\}\} \quad (6)$$

This subset contains only examples from these two classes.
Decision trees (DTs) are trained on a subset of the dataset features. These features are indexed by set $F \subseteq \{1, \dots, d\}$. Decision tree predictions T_F for input samples x are indicated by $T_F(x_F)$ where x_F is feature-subvector restricted to indices in F.
Therefore, it can be said that each pair of classes within the data set generates a set of decision trees where each candidate tree is produced by:

- Random subspace selection: pick a subset of feature indices, F, where the size of F is determined as shown in formula 7:

$$F_{size} = \alpha d \quad (7)$$

Where:
$\alpha \in [0,1]$: percentage of features to be selected.
d: Number of features in dataset D.

- Bootstrap sampling [52]: At this stage, we draw a bootstrap sample $\mathcal{S}$ of the pair-wise data $D_{i,j}$.
- Training each of the generated DTs T_F on $D_{i,j}$. This stage returns a set of candidate detectors $\mathcal{T}_{i,j}$ which shown in formula (8).

$$\mathcal{T}_{i,j} = \{ (T_{F1}, F_1), \dots, (T_{FM}, F_M)\} \quad (8)$$

Where m is the number of trees for each pair (M=2 in this research).

4.4.2.2 Heuristic Filtering Process

This stage constitutes the principal contribution of the HGS-RF model. Unlike standard RF, which retains all trees in $\mathcal{T}_{i,j}$, HGS-RF subjects each candidate detector (T_F, F) to a performance-based filtering process.
The candidate detectors are then filtered, so that a detector is retained only if it reliably recognizes at least one of the two classes (i.e., if it has a high true positive rate in the "self" examples). The procedure is implemented as follows:
For a detector (T_F, F) and class $c \in \{c_i, c_j\}$, the self set $\mathcal{S}_c$ is defined as shown in formula (9).

$$\mathcal{S}_c = \{(x, y) \in D_{i,j} \mid y = c \quad (9)$$

Compute the fraction of self-samples predicted as class c (Accuracy) as shown in formula (10).

$$Acc_{self}(T_F, c) = \frac{\sum_{(x,y)\in\mathcal{S}_c}[T_F(X_F) = c]}{Size\ of\ \mathcal{S}_c} \quad (10)$$

Then the valid detector is determined, c if $Acc_{self}(T_F, c) \geq \tau$, where τ is a threshold (in this research, $\tau = 0.95$). The detector is considered valid if it is valid for at least one of the two classes in the pair. This can be expressed mathematically as shown in formula (11).

$$Keep\ detector\ (T_F, F) \Leftrightarrow \exists c \in \{c_i, c_j\} : Acc_{self}(T_F, c) \geq \tau \quad (11)$$

This process results a set of filtered detectors $\tilde{\mathcal{T}}_{i,j} \in \mathcal{T}_{i,j}$ that achieve high accuracy in detecting one or more pairs of classes. This filtering method reduces spurious detectors that do not reliably recognize their intended class (self), and this prioritizes high true positive coverage for at least one class.

This heuristic filter ensures that the final ensemble is composed exclusively of trees that have proven high proficiency in recognizing at least one specific class of traffic, effectively eliminating weak, non-specialized, or noisy learners. All filtered detectors are aggregated into a final, global detector bank: $\mathcal{B} = \bigcup_{i<j} \tilde{\mathcal{T}}_{i,j}$.

Here, the threshold is defined as follows: τ = 0.95. This threshold is used as a result of the initial experiments. A high threshold is used to ensure that each detector used has nearly perfect recognition of its class ('self'), as in the metaphor of the immune system, where detectors that are not quite so competent do not mature.
Although using a threshold makes the implementation easier, we also recognize that different thresholds may be appropriate for different types of vulnerabilities, a topic which we also discuss in the section on our future work.

4.4.2.3 Ensemble Predictions

The vote $V(c, x)$ for one of the two classes in its pair. The final prediction is the class with the maximum votes, as shown in formula (12).

$$\hat{y}(x) = argmax_{c\in C} V(c, x) \quad (12)$$

4.4.2.4 From Detector Ensemble to Multi-Label Predictions

In the HGS-RF framework, each filtered detector $(T_F, F) \in \mathcal{B}$ is specialized for a specific vulnerability type. Specifically, for each vulnerability type c. We maintain a set of detectors $\mathcal{B}_c \subseteq \mathcal{B}$ that were trained on subsets involving c and passed the filtering threshold for class cc. These detectors serve as experts for identifying that particular vulnerability. For a new contract x, the prediction process proceeds as follows:

- Per-Detector Voting: Each detector $(T_F, F) \in \mathcal{B}_c$ examines the contract and produces a binary prediction: $T_F(x_F) \in \{0,1\}$, where 1 indicates the detector believes vulnerability c is present, and 0 indicates it believes c is absent.
- For each vulnerability type c, we aggregate the votes from all detectors in $\mathcal{B}_c$:

$$V_c(x) = \sum_{(T_F,F)\in\mathcal{B}_c} \mathbb{I}(T_F(x_F) = 1)$$

  Where $\mathbb{I}(.)$ is the indicator function.
- A vulnerability c is predicted to be present in contract x if the proportion of detectors voting positive exceeds a per-vulnerability threshold $\tau_c = 0.5$.

This process is illustrated in Figure 7, which shows how multiple detectors for different vulnerability types can independently identify co-existing vulnerabilities within a single contract.

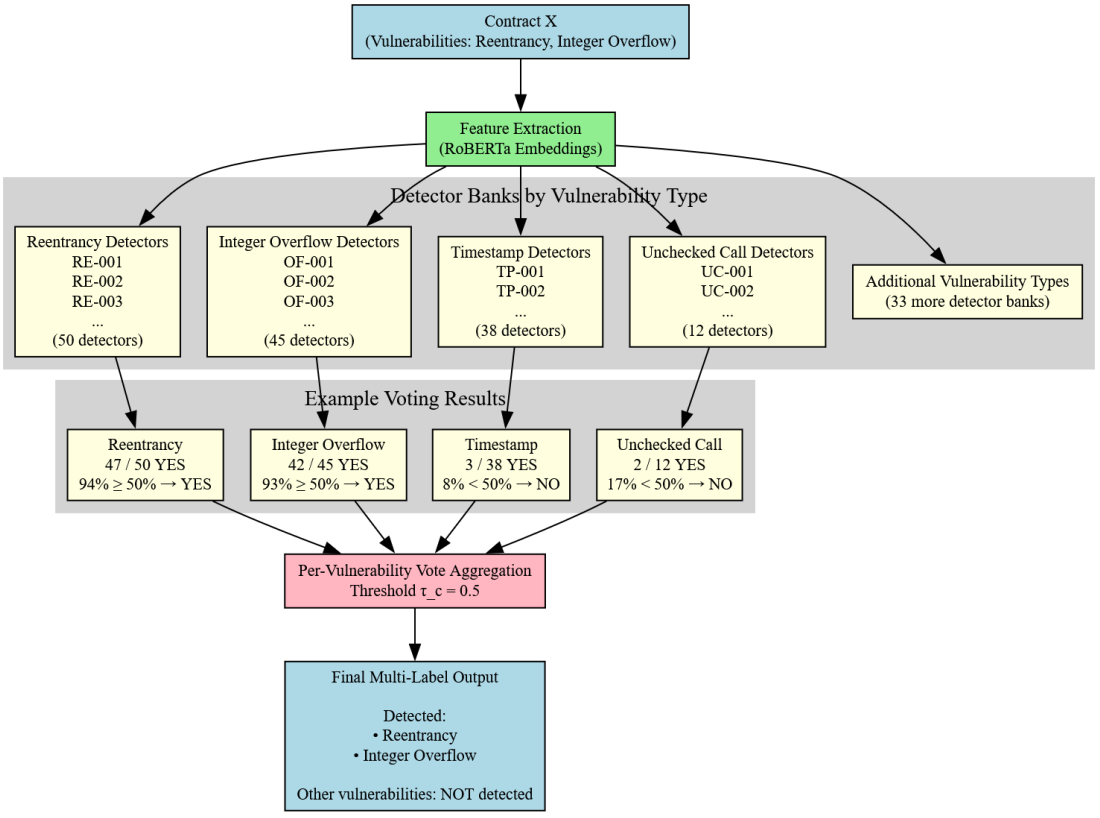

Fig. 7. Multi-label prediction process in HGS-RF. Each vulnerability type has a dedicated bank of specialized detectors. Detectors vote independently, votes are aggregated per type, and a threshold ($\tau_c = 0.5$) determines whether each vulnerability is predicted present. The final output is a set of vulnerability types, enabling detection of multiple co-existing vulnerabilities in a single contract.

4.4.2.5 Workflow and Implementation

Figure 8 shows the workflow diagram of the proposed methodology.

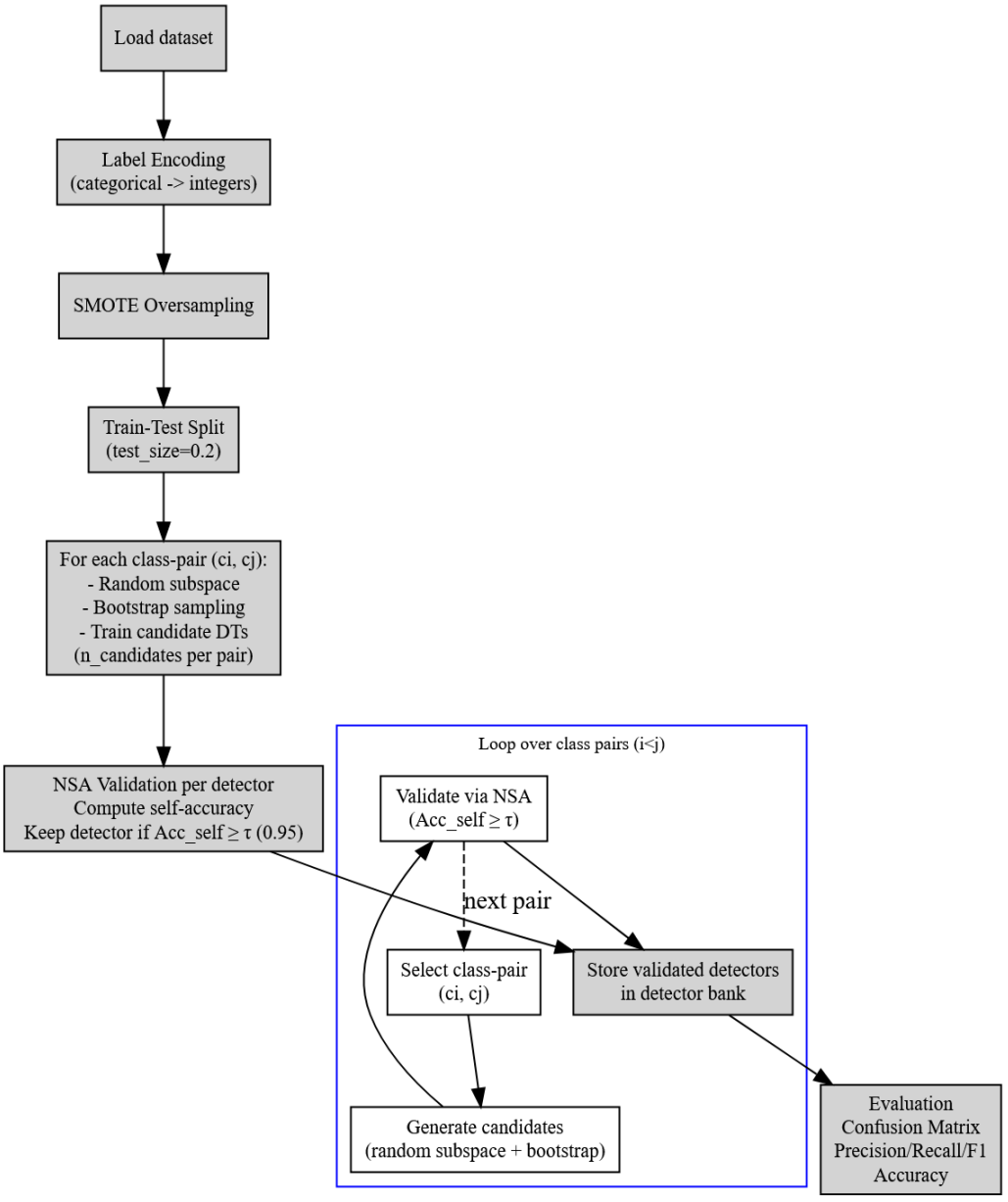

Fig. 8. The workflow diagram of the proposed methodology

The main contribution of this method is that previous multi-category classification methods failed to detect all vulnerabilities because they only produced one type of vulnerability. Many previous research on vulnerabilities in smart contracts has relied on binary and multi-class classification problems, and there is limited work exists have addressed what is called multi-label vulnerabilities, meaning that a single contract contains multiple types of vulnerabilities. The proposed method enables the creation of specialized detectors for each type of vulnerability. A majority vote can then be taken for all detectors together to create a multi-category classification, or a majority vote can be taken for each group of detectors specializing in specific types of vulnerabilities to create a multi-label classification.

4.5 Multi-Label Evaluation Framework

For each node, the model generates a binary vector $\hat{y} \in \{0,1\}^m$ (where m is the number of vulnerability types), which is compared to the real vector $y \in \{0,1\}^m$. A 2x2 confusion matrix is generated for each vulnerability c, showing the number of correct and incorrect predictions for that vulnerability. Key performance metrics are calculated from this binary matrix. Figure 9 shows the structure of the confusion matrix which generated for each vulnerability types.

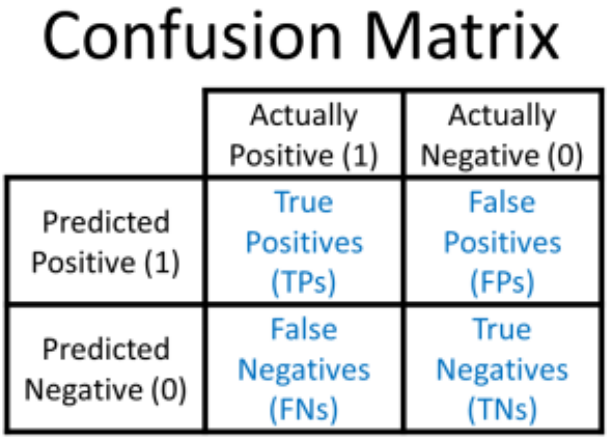

Fig. 9. The structure of confusion matrix

Where:

- True Positives (TP): Number of contracts with vulnerability c and correct prediction
- True Negatives (TN): Number of contracts with no vulnerability c and correct prediction
- False Positives (FP): Number of contracts with no vulnerability c and incorrect prediction
- False Negatives (FN): Number of contracts with vulnerability c and incorrect prediction

The confusion matrix is used to calculate the following performance measures for each type of vulnerabilities.

$$Re\,c\,all = \frac{TP}{TP + FN}$$

$$Pr\,e\,cision = \frac{TP}{TP+FP}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$F1\_Score = 2 \times \frac{Pr\,ecision \times Re\,call}{Pr\,ecision + Re\,call}$$

The overall accuracy refers specifically to subset accuracy (exact match ratio), the proportion of contracts for which all vulnerability predictions exactly match the ground truth. This is the most challenging metric and demonstrates the model's ability to correctly identify complete vulnerability profiles.

4.6 Experimental Setup and Data Splitting

After performing the feature extraction process, we checked for the presence of duplicate records within each data set that was worked on to ensure that no data leakage occurred between the training and test sets. We found that there were no duplicate records within the data set after feature extraction. After checking for duplicate records, we randomly split the dataset into training sets (80%) and a test set (20%). All splits were generated using a fixed random seed (42) to ensure reproducibility. Only the training data was subjected to the Synthetic Minority Oversampling Technique (SMOTE) after being split. The validation and test sets remained untouched by SMOTE to ensure that evaluation reflected performance on real (non-synthetic) data. This ensures that no synthetic data influences validation or test performance, preventing data leakage.

5 RESULTS

All the results reported in this section are based on the hold-out test sets (20% of each dataset) as defined in the splitting procedure described in Section 4.6. The training set and validation set were used for model development and hyperparameter tuning.

5.1 Binary Classification Results

Figure 10 shows the Confusion Matrix (CM) (Dataset 1).

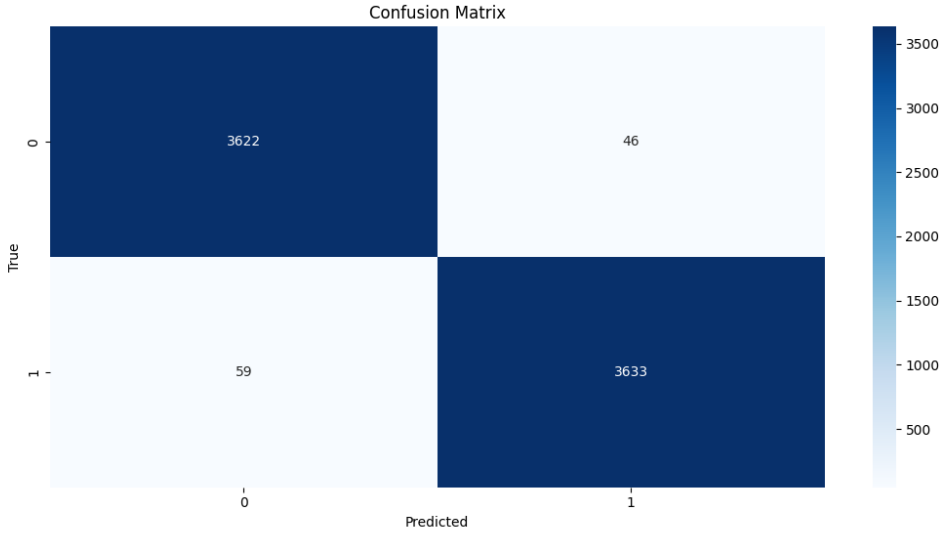

Fig. 10. The CM result (Dataset 1)

Figure 10 shows that the proposed model correctly predicted 3622 class0 samples and 3633 class1 samples. It also misclassified 46 class1 samples but were actually class0, and misclassified 59 class0 but actually class1.

Table 6 shows the performance (Dataset 1).

TABLE VI. THE PERFORMANCE METRICS (DATASET 1)

	Precision	Recall	F1_Score
0	0.98	0.99	0.99
1	0.99	0.98	0.99

From the performance metrics presented in Table 6, it can be observed that the proposed model achieved a precision of 0.98 for class 0, indicating that 0.98 of the instances predicted as class 0 were correctly classified. Similarly, 0.99 of the instances predicted as class 1 were correctly identified. Recall reflects the proportion of correctly predicted samples relative to the actual number of samples in each class. Accordingly, the model correctly identified 0.99 of class 0 instances and 0.98 of class 1 instances. F1 was 0.99 and 0.99 for classes and 1, respectively.
Figure 11 shows a comparison between the baseline algorithms and the proposed model in terms of accuracy (Dataset 1).

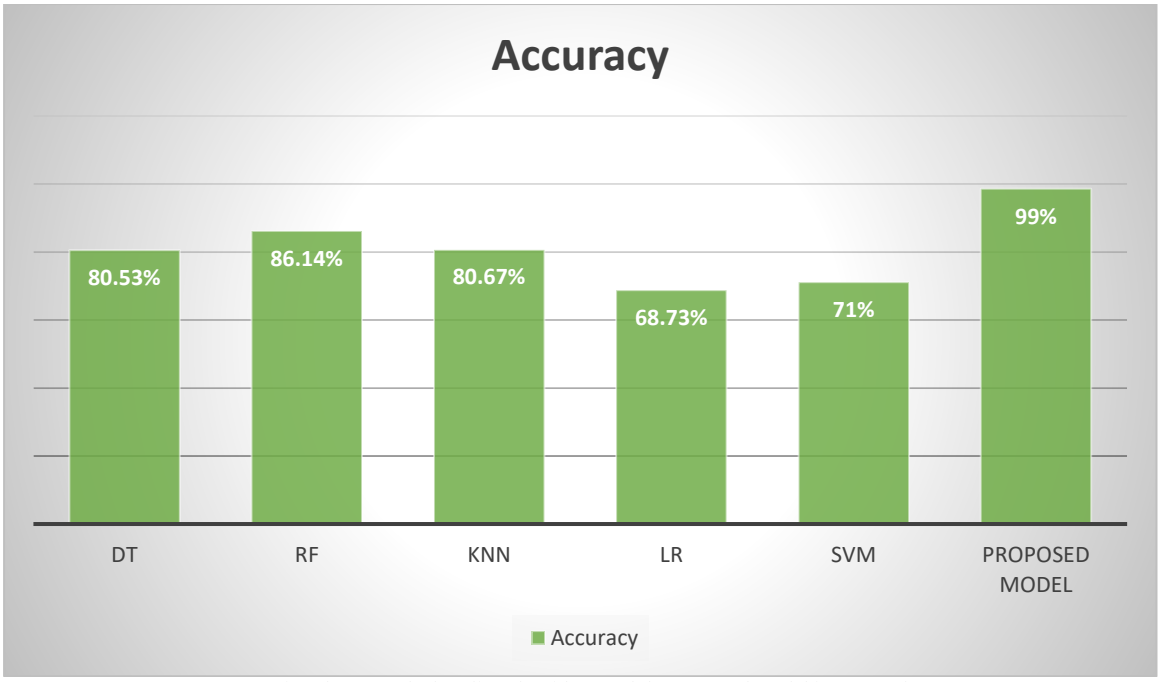

Fig. 11. A comparison between the baseline algorithms and the proposed model in terms of accuracy (Dataset 1)

Figure 11 illustrates the performance comparison of the proposed model HGS-RF with various baseline models. As shown in the figure, the proposed model achieves the best accuracy in the evaluation of the different types of vulnerability. This is due to the hybrid model's capacity to include the performance of the context features learned by the proposed model's RoBERTa model and the ensemble learning of the RF model. The heuristic filtering stage of the proposed model contributes to the improvement of the model's performance by enabling the model to learn from the relevant vulnerability features.

5.2 Multiclass Classification Results (Dataset 2)

Figure 12 shows the CM results (Dataset 2).

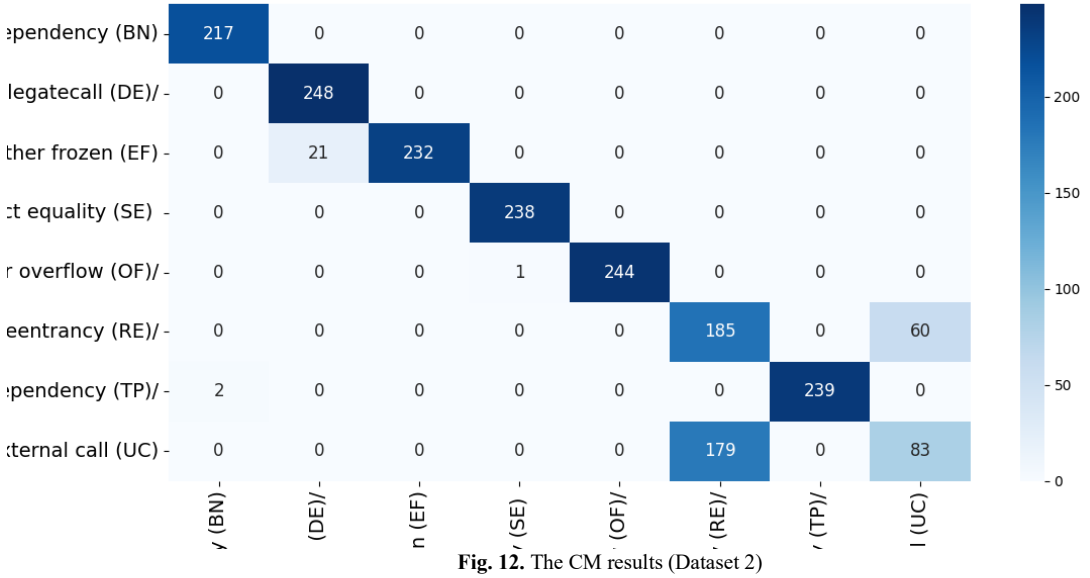

Fig. 12. The CM results (Dataset 2)

The confusion matrix in figure 12 provides a snapshot of the performance of DT algorithm. Here are a few key observations:
The diagonal elements represent the correct predictions for each class [45]. For example, the top-left element (217) indicates 217 correct predictions for BN class.

Off-diagonal elements represent misclassifications. For instance, the element in the third row, second column (21) indicates that 21 instances of EF class were misclassified as DE class.

Table 7 shows the performance metrics (Dataset 2).

TABLE VII. THE PERFORMANCE METRICS (DATASET 2)

	Precision	Recall	F1_Score
BN	0.99	1.00	0.99
DE	0.92	1.00	0.95
EF	1.00	0.91	0.95
SE	0.99	1.00	0.99
OF	1.00	0.99	0.99
RE	0.50	0.75	0.60
TP	1.00	0.99	0.99
UC	0.58	0.31	0.41

Figure 13 shows a comparison between the baseline algorithms and the proposed model in terms of accuracy (Dataset 2).

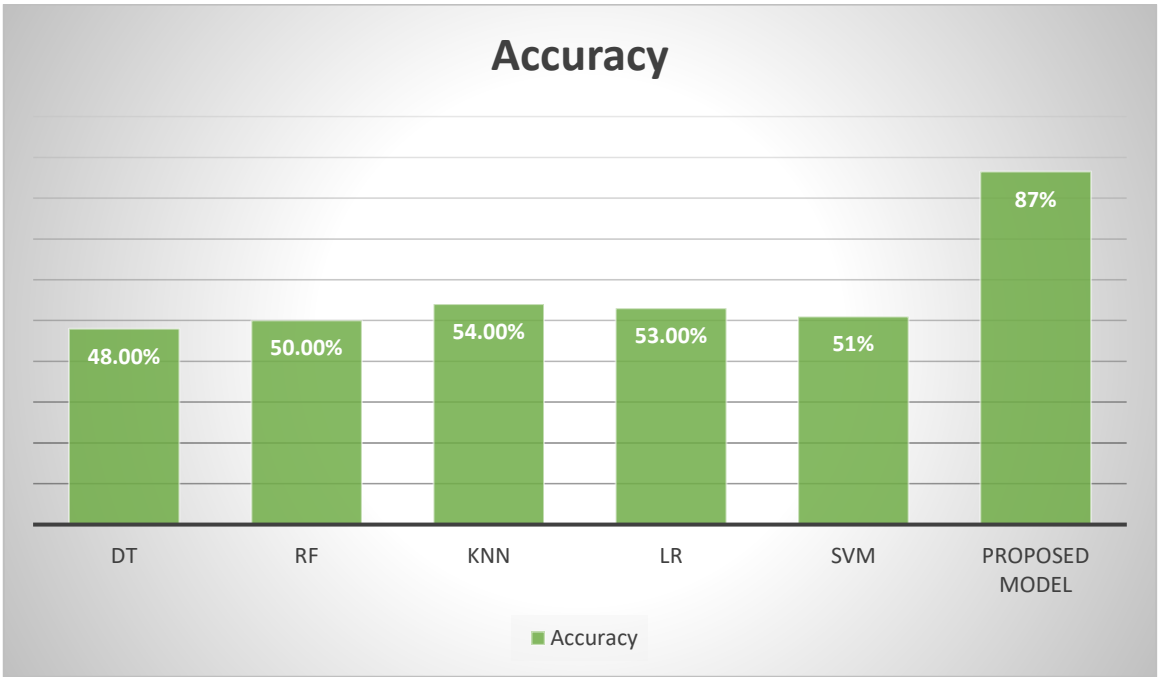

Fig. 13. A comparison between the baseline algorithms and the proposed model in terms of accuracy (Dataset 2)

Figure 11 illustrates the performance comparison of the proposed model HGS-RF with various baseline models. As shown in the figure, the proposed model achieves the best accuracy in the evaluation of the different types of vulnerability.

5.3 Multiclass Classification Results (Dataset 3)

Figure 14 shows the CM results (Dataset 3).

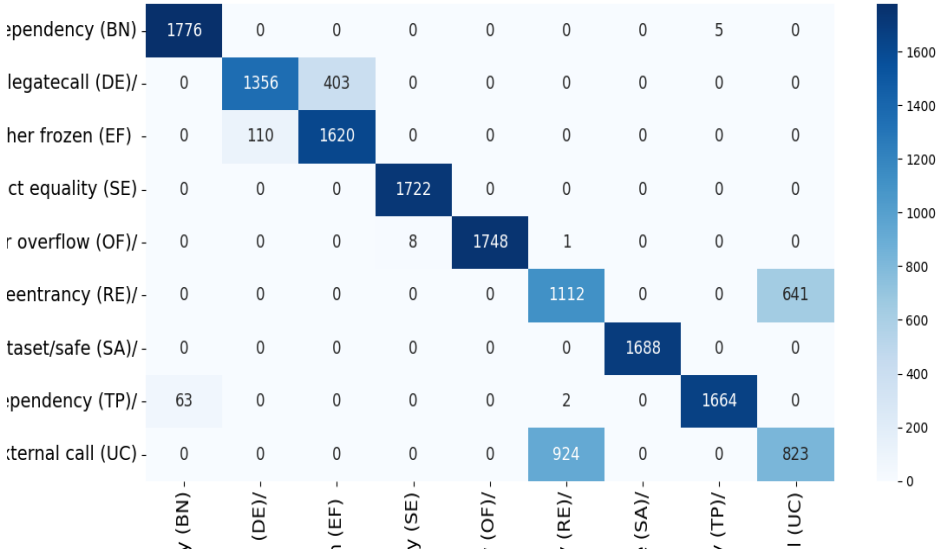

Fig. 14. The CM results (Dataset 3)

Table 8 shows the performance metrics (Dataset 3).

TABLE VIII. THE PERFORMANCE METRICS (DATASET 3)

	Precision	Recall	F1_Score
BN	0.96	0.99	0.98
DE	0.92	0.77	0.84
EF	0.80	0.93	0.86
SE	0.99	1.00	0.99
OF	1.00	0.99	0.99
RE	0.54	0.63	0.58
SA	1.00	1.00	1.00
TP	0.99	0.96	0.97
UC	0.56	0.47	0.51

Figure 15 shows a comparison between the baseline algorithms and the proposed model in terms of accuracy (Dataset 3).

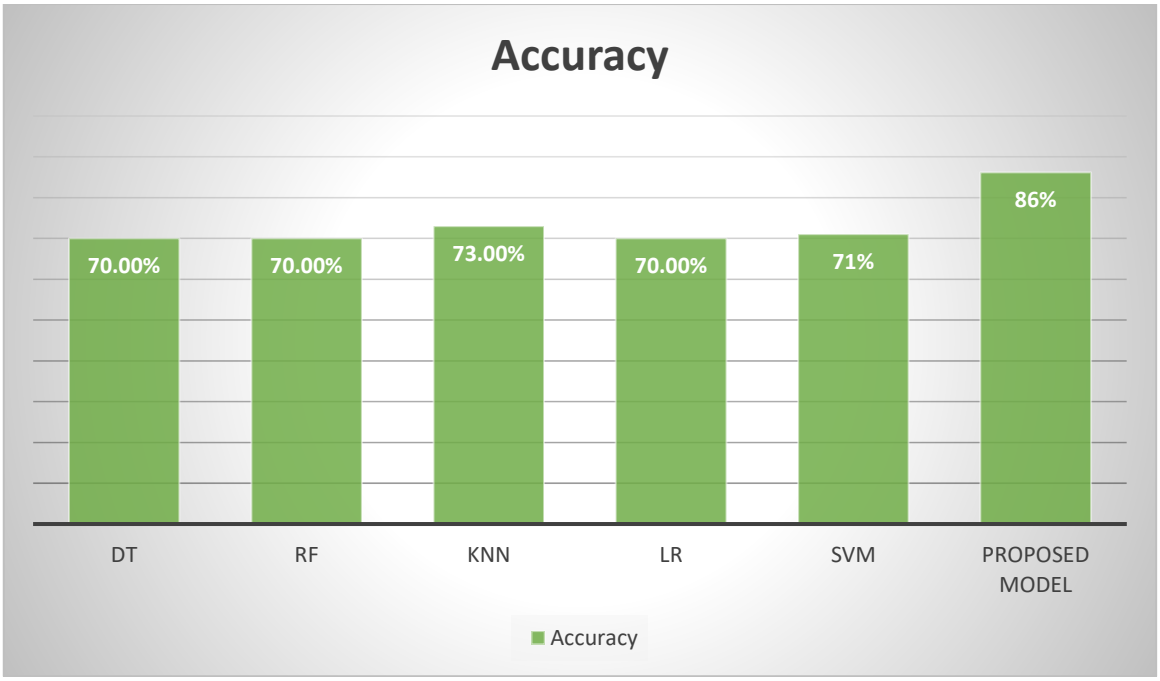

Fig. 15. A comparison between the baseline algorithms and the proposed model in terms of accuracy (Dataset 3)

5.4 Multilabel Detection in Smart Contracts (Dataset 4)

This research contributed to the development of a multi-label classification approach with significant imbalances in its categories, reflecting the distribution of security vulnerabilities in the real world.
Dataset Characteristics:

- Total Contracts: 15,000
- Vulnerability Types: 34

The classification relies on generating decision tree classifiers for each vulnerability type and then grouping the decisions made by each classifier according to the vulnerability it is responsible for. This results in multiple decisions that identify the various vulnerabilities present within the node. The model proposed in this research contribution is superior because all previous studies rely either on a binary classification that categorizes nodes as either healthy or vulnerable, or a multi-category classification that identifies the vulnerability type present in each node individually. However, if a node contains multiple vulnerabilities, the current methods fail to detect them all; only one type is identified in the output of the classifiers used in previous studies.

Table 9 shows the resulting CM for each type of vulnerability (Dataset 4).

TABLE IX. THE RESULTING CM FOR EACH TYPE OF VULNERABILITY (DATASET 4)

	Safe		arbitrary-send		backdoor		boolean-cst	
0	2100	4	2719	0	3000	0	2998	0
1	14	882	7	274	0	0	0	2
	constant-function-asm		constant-function-state		controlled-array-length		controlled-delegatecall	
0	2915	0	2994	0	2909	2	2958	0
1	0	85	0	6	4	85	0	42
	delegatecall-loop		divide-before-multiply		erc20-interface		erc721-interface	
0	2999	0	2503	5	2862	2	2997	0
1	0	1	14	478	3	133	1	2
	incorrect-equality		incorrect-shift		locked-ether		mapping-deletion	
0	2648	4	2973	0	2794	0	2994	0
1	15	333	1	26	3	203	0	6
	msg-value-loop		reentrancy-eth		reentrancy-no-eth		shadowing-abstract	
0	2998	0	2489	0	2544	6	2972	0

1	0	2	3	508	25	425	0	28
	shadowing-state		suicidal		tautology		tx-origin	
0	2938	1	2996	0	2884	0	2904	0
1	1	60	0	4	2	114	3	93
	unchecked-lowlevel		unchecked-send		unchecked-transfer		uninitialized-local	
0	2948	0	2992	0	2671	3	2631	2
1	0	52	0	8	12	314	12	355
	uninitialized-state		uninitialized-storage		unprotected-upgrade		unused-return	
0	2871	0	2999	0	2992	0	2060	2
1	5	124	0	1	0	8	8	930
	weak-prng		write-after-write					
0	2939	0	2970	1				
1	2	59	0	29				

Table 10 shows the resulting performance metrics for each type of vulnerability (Dataset4).

TABLE X. THE RESULTING PERFORMANCE METRICS FOR EACH TYPE OF VULNERABILITY (DATASET4)

	Safe			arbitrary-send			backdoor			boolean-cst		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	0.993	0.998	0.996	0.997	1.00	0.999	1.00	1.00	1.00	1.00	1.00	1.00
1	0.995	0.984	0.990	1.00	0.975	0.987	-	-	-	1.00	1.00	1.00
	constant-function-asm			constant-function-state			controlled-array-length			controlled-delegatecall		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	1.00	1.00	1.00	0.999	0.999	0.999	1.00	1.00	1.00
1	1.00	1.00	1.00	1.00	1.00	1.00	0.977	0.955	0.966	1.00	1.00	1.00
	delegatecall-loop			divide-before-multiply			erc20-interface			erc721-interface		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	0.994	0.998	0.996	0.999	0.999	0.999	1.00	1.00	1.00
1	1.00	1.00	1.00	0.990	0.972	0.981	0.985	0.978	0.982	1.00	0.667	0.80
	incorrect-equality			incorrect-shift			locked-ether			mapping-deletion		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	0.994	0.998	0.996	1.00	1.00	1.00	0.999	1.00	0.999	1.00	1.00	1.00
1	0.988	0.957	0.972	1.00	0.963	0.981	1.00	0.985	0.993	1.00	1.00	1.00
	msg-value-loop			reentrancy-eth			reentrancy-no-eth			shadowing-abstract		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	0.999	1.00	0.999	0.990	0.998	0.994	1.00	1.00	1.00
1	1.00	1.00	1.00	1.00	0.994	0.997	0.986	0.944	0.965	1.00	1.00	1.00
	shadowing-state			suicidal			tautology			tx-origin		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	1.00	1.00	1.00	0.999	1.00	1.00	0.999	1.00	0.999
1	0.984	0.984	0.984	1.00	1.00	1.00	1.00	0.983	0.991	1.00	0.969	0.984
	unchecked-lowlevel			unchecked-send			unchecked-transfer			uninitialized-local		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	1.00	1.00	1.00	0.996	0.999	0.997	0.995	0.999	0.997
1	1.00	1.00	1.00	1.00	1.00	1.00	0.991	0.963	0.977	0.994	0.967	0.981
	uninitialized-state			uninitialized-storage			unprotected-upgrade			unused-return		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	0.998	1.00	0.999	1.00	1.00	1.00	1.00	1.00	1.00	0.996	0.999	0.998
1	1.00	0.961	0.980	1.00	1.00	1.00	1.00	1.00	1.00	0.998	0.991	0.995
	weak-prng			write-after-write								
	precision	recall	F1	precision	recall	F1						
0	0.999	1.00	1.00	1.00	1.00	1.00						
1	1.00	0.967	0.983	0.967	1.00	0.983						

In addition, because of the extremely unbalanced nature of the smart contract vulnerability dataset, there may exist some types of vulnerabilities that have very few or no positive samples within the test set. In that situation, the precision or recall of that specific class may not be well defined because the model may not make any predictions for that class or may not have any ground truth samples available. In the per label metric table, a dash (-) is used to represent undefined values. When computing the macro-averaged values, labels with undefined values are excluded from the macro-average computation process to avoid any potential inflation or deflation of the performance scores.

Table 11 shows the overall performance (Dataset 4).

TABLE XI. THE OVERALL PERFORMANCE (DATASET4)

Precision	Recall	F1_Score	Macro F1-score	Micro F1-score	Accuracy
0.994	0.977	0.985	0.955	0.985	0.97

5.5 Multilabel Detection in Smart Contracts (Dataset 5)

Dataset Characteristics:

- Total Contracts: 15,000
- Vulnerability Types: 37

Table 12 shows the resulting CM for each type of vulnerability (Dataset 5).

TABLE XII. THE RESULTING CM FOR EACH TYPE OF VULNERABILITY (DATASET5)

	Safe		arbitrary-send		array-by-reference		boolean-cst	
0	2043	5	2714	0	3000	0	2996	0
1	16	936	5	281	0	0	0	4
	constant-function-asm		constant-function-state		controlled-array-length		controlled-delegatecall	
0	2926	0	2998	0	2912	0	2940	0
1	0	74	0	2	1	87	5	55
	delegatecall-loop		divide-before-multiply		erc20-interface		erc721-interface	
0	2997	0	2478	1	2875	0	2999	0
1	1	2	18	503	2	123	0	1
	incorrect-equality		incorrect-shift		locked-ether		mapping-deletion	
0	2613	2	2976	0	2814	1	2992	0
1	16	369	0	24	6	179	1	7
	msg-value-loop		name-reused		public-mappings-nested		reentrancy-eth	
0	2998	0	2981	0	3000	0	2465	0
1	0	2	2	17	0	0	5	530
	reentrancy-no-eth		reused-constructor		shadowing-abstract		shadowing-state	
0	2570	0	3000	0	2968	0	2947	0
1	15	415	0	0	0	32	6	47
	suicidal		tautology		tx-origin		unchecked-lowlevel	
0	2993	0	2878	0	2920	3	2944	0
1	0	7	1	121	0	77	0	56
	unchecked-send		unchecked-transfer		uninitialized-local		uninitialized-state	
0	2991	0	2704	2	2610	2	2855	0
1	0	9	5	289	5	383	10	135
	uninitialized-storage		unprotected-upgrade		unused-return		weak-prng	
0	2999	0	2997	0	2000	2	2936	1
1	0	1	0	3	13	985	1	62
	write-after-write							
0	2948	0						
1	1	51						

Table 13 shows the resulting performance metrics for each type of vulnerability (Dataset 5).

TABLE XIII. THE RESULTING PERFORMANCE METRICS FOR EACH TYPE OF VULNERABILITY (DATASET5)

	Safe			arbitrary-send			array-by-reference			boolean-cst		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	0.992	0.998	0.995	0.998	1.00	0.999	1.00	1.00	1.00	1.00	1.00	1.00
1	0.995	0.983	0.989	1.00	0.983	0.991	-	-	-	1.00	1.00	1.00
	constant-function-asm			constant-function-state			controlled-array-length			controlled-delegatecall		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.998	1.00	0.999

1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.989	0.994	1.00	0.917	0.957
	delegatecall-loop			divide-before-multiply			erc20-interface			erc721-interface		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	0.993	1.00	0.996	0.999	1.00	1.00	1.00	1.00	1.00
1	1.00	0.667	0.80	0.998	0.965	0.981	1.00	0.984	0.992	1.00	1.00	1.00
	incorrect-equality			incorrect-shift			locked-ether			mapping-deletion		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	0.994	0.999	0.997	1.00	1.00	1.00	0.998	1.00	0.999	1.00	1.00	1.00
1	0.995	0.958	0.976	1.00	1.00	1.00	0.994	0.968	0.981	1.00	0.875	0.933
	msg-value-loop			name-reused			public-mappings-nested			reentrancy-eth		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	0.999	1.00	1.00	1.00	1.00	1.00	0.998	1.00	0.999
1	1.00	1.00	1.00	1.00	0.895	0.944	-	-	-	1.00	0.991	0.995
	reentrancy-no-eth			reused-constructor			shadowing-abstract			shadowing-state		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	0.994	1.00	0.997	1.00	1.00	1.00	1.00	1.00	1.00	0.998	1.00	0.999
1	1.00	0.965	0.982	-	-	-	1.00	1.00	1.00	1.00	0.887	0.940
	suicidal			tautology			tx-origin			unchecked-lowlevel		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.999	0.999	1.00	1.00	1.00
1	1.00	1.00	1.00	1.00	0.992	0.996	0.963	1.00	0.981	1.00	1.00	1.00
	unchecked-send			unchecked-transfer			uninitialized-local			uninitialized-state		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	0.998	0.999	0.999	0.998	0.999	0.999	0.997	1.00	0.998
1	1.00	1.00	1.00	0.993	0.983	0.988	0.995	0.987	0.991	1.00	0.931	0.964
	uninitialized-storage			unprotected-upgrade			unused-return			weak-prng		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
0	1.00	1.00	1.00	1.00	1.00	1.00	0.994	0.999	0.996	1.00	1.00	1.00
1	1.00	1.00	1.00	1.00	1.00	1.00	0.998	0.987	0.992	0.984	0.984	0.984
	write-after-write											
	precision	recall	F1									
0	1.00	1.00	1.00									
1	1.00	0.981	0.990									

In addition, because of the extremely unbalanced nature of the smart contract vulnerability dataset, there may exist some types of vulnerabilities that have very few or no positive samples within the test set. In that situation, the precision or recall of that specific class may not be well defined because the model may not make any predictions for that class or may not have any ground truth samples available. In the per label metric table, a dash (-) is used to represent undefined values. When computing the macro-averaged values, labels with undefined values are excluded from the macro-average computation process to avoid any potential inflation or deflation of the performance scores.

Table 14 shows the overall performance (Dataset 5).

TABLE XIV. THE OVERALL PERFORMANCE (DATASET5)

Precision	Recall	F1_Score	Macro F1-score	Micro F1-score	Accuracy
0.997	0.978	0.987	0.901	0.987	0.967

5.6 Threshold Sensitivity Analysis (Dataset 5):

In order to test the effectiveness of fixed retention threshold value τ, sensitivity analysis was carried out by varying the value of τ between 0.80 and 0.99 on the validation set of Dataset 5. Table 15 depicts the relationship between threshold values and performance metrics.

TABLE XV. THE RELATIONSHIP BETWEEN THRESHOLD VALUES AND PERFORMANCE METRICS

τ	Precision	Recall	F1_Score
0.8	0.932	0.985	0.958
0.82	0.941	0.984	0.962
0.84	0.95	0.984	0.967
0.86	0.959	0.983	0.971
0.88	0.967	0.983	0.975

0.9	0.975	0.982	0.978
0.91	0.981	0.981	0.981
0.92	0.986	0.981	0.983
0.93	0.99	0.98	0.985
0.94	0.993	0.979	0.986
0.95	0.997	0.978	0.987
0.96	0.997	0.972	0.984
0.97	No detectors	No detectors	No detectors
0.98	No detectors	No detectors	No detectors
0.99	No detectors	No detectors	No detectors

The F1-score achieves its maximum value of 0.987 at $\tau = 0.95$, with precision 0.997 and recall 0.978. This represents the optimal balance between detector quality and ensemble diversity.

For values of τ less than or equal to 0.90, the model shows obvious signs of under-filtering. For instance, for a value of $\tau = 0.80$, the precision is only 0.932, a loss of 6.5% as opposed to the optimal value. This is due to the fact that weak detectors, which are more likely to be noisy, are incorporated in the ensemble. This leads to incorrect positive predictions. The high values of recall, ranging from 0.982 to 0.985, for values of τ less than 0.90 are due to the diversity of the detectors incorporated in the ensemble. The enhancement in the values of precision from 0.932 to 0.975 for increasing values of τ from 0.80 to 0.90 indicates the effect of the removal of weak detectors.

6 DISCUSSIONS

6.1 Binary classification results

We note that traditional ML algorithms showed severe limitations in detecting vulnerable contracts, as shown in Table 16, which represents a summary of conventional algorithms' performance. The large performance difference, particularly for class 1 (vulnerable contracts), reveals a big flaw in traditional ML algorithms. These algorithms have a strong bias to the majority class (healthy contracts), which leads to a dangerously high false negative rate. This scenario is very dangerous because it enables catastrophic financial losses in real-world blockchain applications. The binary classification results presented in Section 5.1 reveal some very interesting trends that not only shed more light on the quality of the suggested approach but also on the inherent characteristics of the problem of detecting vulnerabilities. This proves the point that there is a limitation of the conventional algorithms, and the limitation is based on the bias of the conventional algorithms towards the majority class. There is a ratio of about 3 healthy contracts to 1 vulnerable contract in the Dataset 1. The conventional algorithms have the goal of optimizing the accuracy of all the contracts. However, the only way to achieve the goal of optimizing the accuracy of all the contracts is to predict the majority class most of the time, and this leads to unacceptably high false negatives for the vulnerable contracts. False negatives have devastating financial consequences for the real-world applications of the blockchain technology. The excellent performance of our approach can be attributed to several factors:

- Prototype-based representation: By learning the representation where each class is represented by a prototype (which is the average of the support samples in the class). The decision boundary is naturally in terms of a distance to a prototype rather than a decision hyperplane learned from the data that could be biased due to the class distribution.
- Metric learning objective: The objective of training (i.e., minimizing the distance between the query samples and their respective classes' prototypes) ensures the robustness of the model against any change in the distribution of classes.
- Few Shot Formulation: By taking balanced "shots" during training, the model is never exposed to the imbalanced distribution during training, even though the underlying dataset is imbalanced.
- Dimensionality Reduction: In the proposed MLP, the dimensionality is reduced from 768 to 64 features. This is done to ensure that only the most discriminative features are utilized without overfitting to spurious features.

TABLE XVI. SUMMARY OF TRADITIONAL ALGORITHMS PERFORMANCE

Algorithm	Class 0 F1	Class 1 F1	Accuracy
DT	0.84	0.52	~0.78
RF	0.88	0.53	~0.82
KNN	0.86	0.55	~0.80
LR	0.86	0.45	~0.79
SVM	0.86	0.46	~0.79

The architectural superiority in binary classification resulted from the few-shot learning advantage, which leads to exceptional performance (F1-scores of 0.98-0.99) compared to traditional algorithms (F1-scores of 0.45-0.55 for the

vulnerable class). This is due to its innovative use of Prototypical Networks within a few-shot learning framework. This approach learns semantic representations and creates prototype embeddings that capture the essential characteristics of "vulnerable" and "healthy" contract semantics. The main advantage of this approach is that the prototype-based classification naturally accommodates imbalanced distributions without the need to create synthetic samples. The Euclidean distance measurement in the reduced embedding space (64 dimensions vs. original 768) creates a more robust and generalizable decision mechanism than the probability estimates of conventional classifiers.

6.2 Multi-Class Classification Results

Table 17 shows the multi-class performance level for multi-class classification results.

TABLE XVII. THE MULTI-CLASS PERFORMANCE LEVEL FOR DATASET 2

Vulnerability Type	Precision	Recall	F1-Score	Performance Level
TP	1.00	0.99	0.99	High
EF	1.00	0.91	0.95	High
OF	1.00	0.99	0.99	High
DE	0.92	1.00	0.95	High
BN	0.99	1.00	0.99	High
SE	0.99	1.00	0.99	High
RE	0.50	0.75	0.60	Medium
UC	0.58	0.31	0.41	Low

We note from Table 17 that the performance level suggests that vulnerabilities are segmented into distinct categories based on their detectability. (1) The vulnerabilities with blue color represent high-detectability vulnerabilities. We can explain this to its consistent, localized code patterns with strong semantic signatures. (2) The vulnerabilities with green color represent medium-detectability vulnerabilities that may have more complex interactions or diverse implementation patterns. (3) The vulnerabilities with red color represent low-detectability vulnerabilities that require an understanding of cross-contract interactions or exhibit highly variable code patterns. The multi-class results on Datasets 2 and 3 (Sections 5.2-5.3) provide a hierarchy of detectability of vulnerabilities, which gives insight into the behavior of the model as well as into the nature of the vulnerabilities.

- High Detectability Vulnerabilities (F1 $\geq$ 0.95)

Vulnerabilities like Timestamp Dependency (TP), Integer Overflow (OF), Block Number Dependency (BN), etc., have high consistency in their code patterns. For example:

- Timestamp Dependency: Almost always involves code where block.timestamp or now is used directly in a conditional statement
- Integer Overflow: Almost always involves arithmetic operations in the absence of bound checks, following a predictable pattern
- Strict Equality (SE): Almost always involves require(balance == amount), which is easy to identify

These vulnerabilities have high semantic signal strength. These vulnerabilities can easily be identified through the use of contextual embeddings from the RoBERTa model. The code patterns for these vulnerabilities are limited in scope (local to a function) and do not vary significantly across contracts.

- Medium Detectability Vulnerabilities (F1 $\approx$ 0.60):

The reentrancy problem (RE) is a complex problem. The basic reentrancy problem is well understood, the external call before the update of the state. However, the reality of the contracts is different. Indeed, the reentrancy problem may include the following features:

- Several external calls may be included in the reentrancy problem
- Several state update operations may be included in the reentrancy problem
- Several cross-contract calls may be included in the reentrancy problem
- Various protection mechanisms may be included in the reentrancy problem

The average results of the method for the reentrancy problem (F1 = 0.60 for Dataset 2, F1 = 0.58 for Dataset 3) are caused by the complexity of the reentrancy problem. Indeed, the static analysis of the source code cannot always identify the reentrancy problem, depending on the order of the cross-contract calls.

- Low Detectability Vulnerabilities (F1 < 0.60)

The hardest type of vulnerability to manage was found to be the Unchecked External Calls type, with the best F1 scores being 0.41 on Dataset 2 and 0.51 on Dataset 3. By examining the misclassified contracts for the Unchecked External Calls type, the following causes of the problems were determined:

- Unchecked external calls can be subject to conditional checks that are semantically but not syntactically similar to the correct checks.
- Some contracts can involve the usage of low-level calls like call(), delegatecall() for which the handling of return values is more complex.
- Some considerations related to the context of the call can be relevant for the detection of the given type of vulnerability, for instance, the relevance of the return value for the following code flow.

The moderate performance on reentrancy detection warrants particular attention because these attacks represent one of the most devastating vulnerabilities in smart contracts.

TABLE XVIII. THE PERFORMANCE COMPARISON FOR DATASET2 AND DATASET3

Vulnerability	Dataset 2 F1	Dataset 3 F1	Performance Trend
BN	0.99	0.98	Consistent
DE	0.95	0.84	Slight decrease
EF	0.95	0.86	Slight decrease
SE	0.99	0.99	Consistent
OF	0.99	0.99	Consistent
RE	0.60	0.58	Consistent
TP	1.00	0.97	Consistent
UC	0.58	0.51	Slight decrease

As we can see from Table 18, there is a comparison between the performance on Datasets 2 and 3. As we can see, the correlation between the F1 scores is very high, and we can say that we can use this as evidence that we are not learning any kind of artifact specific to a given dataset but learning general characteristics of the vulnerabilities. The consistency across the two datasets indicates that the model learns the main characteristics of the vulnerabilities. The performance differences in some vulnerabilities (DE, EF, UC) reflect differences in how these vulnerabilities showed across different contracts. As we can see from the performance on DE, EF, and UC in Dataset 3, we can say that the performance has dropped slightly because of the number of contracts within the dataset, as they express more diversity with regards to the vulnerabilities. This is a very good sign because we can see that when we are given more diversity within a dataset, we do not perform catastrophically worse but degrade gracefully. These findings also have considerable implications for the wider space of smart contract vulnerability detection. These implications include the following:

- Reentrancy is an issue: Despite being one of the most studied vulnerabilities, reentrancy detection remains non-trivial, suggesting the need for hybrid approaches combining static analysis with dynamic or formal methods.
- Semantic information is useful but not sufficient: Despite the fact that the information provided by the embeddings of the RoBERTa model seems to be useful, there might be specific scenarios of vulnerabilities with cross-contract interactions or complex data flow may require additional structural information.

6.3 Multilabel Classification Results

The results obtained for the multi-label problem with Datasets 4 and 5 (Sections 5.4-5.5) represent the main contribution of this research, as they prove the ability of the HGS-RF model to successfully identify multiple co-existing vulnerabilities within a single contract. This section will provide an in-depth analysis of the results obtained for the model's success.

The subset accuracy (exact match ratio) of 0.97 (Dataset 4) and 0.967 (Dataset 5) is noteworthy. This metric demands that the model be capable of predicting the entire vulnerability profile of a given contract with absolute precision—in the sense that all the vulnerabilities must be predicted as well as all the vulnerabilities that are not part of the given contract must be predicted with equal accuracy. To appreciate the stringency:

- If a given contract has 3 types of vulnerabilities, the model must be capable of predicting all 3 types of vulnerabilities while not predicting any of the other 31-34 types of vulnerabilities
- Not a single miscalculation or misprediction is allowed for the entire contract to be classified as mis predicted by the model for the calculation of the subset accuracy metric
- The ability to achieve a high level of precision greater than 96% for 34-37 types of labels suggests not only the ability to accurately predict the types of vulnerabilities for a given contract but also the ability to accurately determine the entire vulnerability profile for the vast majority of contracts.

Some of the special features that have contributed to the success of the HGS-RF are as follows:

- Specialized Detector Banks: By the usage of separate detector banks for different kinds of vulnerability detection, the possibility of interference between the different kinds of vulnerability detection is completely eliminated. This would have occurred if the same kind of detector was forced to detect all the different kinds of vulnerability. By the usage of the proposed system, the different detectors are specialized in the detection of different kinds of vulnerability on the basis of the patterns related to the different kinds of vulnerability.
- Heuristic Filtering (Inspired by Immune System): By the usage of the heuristic filtering mechanism, only the detectors with high accuracy related to the 'self' class are allowed to pass. This would have resulted in the occurrence of false positive values. Furthermore, through the sensitivity analysis, it has been shown that the importance of the heuristic filtering mechanism lies in the fact that the precision of the proposed scheme would be compromised to a great extent if the heuristic filtering mechanism was not employed ($\tau < 0.90$).
- Independent Per-Type Voting: This ensures that different types of vulnerabilities are predicted at once for a given contract. It is different from the multi-class approach, where a single label is predicted at a time.
- Majority Voting with Threshold: Majority voting with a threshold of 0.5 ensures the robustness of this model. This is because a vulnerability will be predicted only if a majority agrees on it.

6.4 Model Scalability

The methodology demonstrates scalability in two critical dimensions:

a. Vertical Scalability: Handling increasing dataset sizes without performance degradation.
b. Horizontal Scalability: Accommodating growing numbers of vulnerability types (from 8 to 37) while maintaining detection quality.

The modular design of the proposed system provides scalability:

i. Feature extraction through transformers is parallelizable.
ii. The HGS-RF approach naturally accommodates new vulnerability types through additional specialized detectors.
iii. The few-shot learning framework reduces retraining requirements for new vulnerability classes.

7. CONCLUSIONS AND FUTURE WORKS

This research worked on the detection of vulnerabilities in smart contracts. We proposed a hybrid artificial intelligence framework that combines Natural Language Processing (NLP) models for feature extraction with specialized ensemble learning architectures for classification. The evaluation showed several key findings. First, in binary classification, the model achieved performance (0.98-0.99 F1-score), performing better than traditional algorithms, which showed bias and high false-negative rates. Second, the multi-class classification results showed differences in the performance, where the model successes at detecting vulnerabilities with strong, localized code patterns (e.g., Integer Overflow, Timestamp Dependency at 0.97-0.99 F1-score) while showing a range of improvement on more complex vulnerabilities like Reentrancy. Third, the research broke new ground by tackling multi-label classification, achieving an accuracy (0.967-0.97). This model provides a tool for rapid, and automated security analysis. This model reduces the time, cost, and human error of manual code reviews. The ability to perform multi-label detection is particularly transformative because it simulates the complexity of real-world contracts that often contain multiple vulnerabilities. The main limitation of this research is in the performance on complex vulnerabilities, Reentrancy, and Unchecked External Calls. This suggests that static semantic analysis alone may be insufficient to fully capture vulnerabilities that depend on dynamic runtime states or complex contract interactions. The computational resources required for feature extraction using a transformer may present challenges in deployment in environments that have limited resources.
One limitation of the current study is its exclusive focus on static analysis of source code through semantic feature extraction. Although there are many advantages in using the semantic feature extraction technique, there are cases

where the technique might fail to identify the vulnerabilities. Such cases will have their vulnerabilities during runtime. Such cases will also depend on the interactions of the contracts. Static analysis and dynamic analysis are considered to be complementary rather than competitive. The static analysis provides complete coverage of the code paths without any runtime overhead, whereas the dynamic analysis focuses on runtime behavior and environmental factors. The future works shoukd be examine the possibility of combining the HGS-RF method with dynamic analysis tools such as symbolic analysis or fuzzing, along with bytecode-based feature extraction. The features are also to be expanded by incorporating the dynamic indicators as secondary features of the static method. Another area that shows a possibility of improvement in the future is the establishment of an adaptive threshold for the heuristic filtering. Although the threshold, $\tau = 0.95$, is validated by conducting a sensitivity analysis, it may not be the optimal threshold for all vulnerabilities. For example, vulnerabilities with high semantic variability (e.g., reentrancy) might benefit from lower thresholds to retain diverse detectors, while those with consistent patterns could tolerate higher thresholds. The other area for further research includes optimizing the computational efficiency of the feature extraction process. The current implementation uses the full RoBERTa model for feature extraction. While the full RoBERTa model has high accuracy, it also has high computational requirements. There are many ways for optimizing the current implementation: (1) In scenarios where computational resources are a problem, the usage of the Distil RoBERTa model with static contextual embeddings like Word2Vec and FastText can also be explored, but at the expense of accuracy for the detection task. (2) Quantization and Pruning: Post-training quantization, i.e., quantizing the model from 32 bits to 8 bits, and pruning the weights can potentially reduce memory usage and computation time with negligible accuracy loss.

Conflicts of Interest

The authors declare no conflict of interest.

Funding

This research received no external funding.

Acknowledgment

None.

References

[1] M. Azmat and E. Thanou, “Blockchain-enabled smart contract architecture in supply chain design,” in *Blockchain Driven Supply Chain Management: A Multi-Dimensional Perspective*. Springer, 2023, pp. 1–14.

[2] S. Dong, K. Abbas, M. Li, and J. Kamruzzaman, “Blockchain technology and application: An overview,” *PeerJ Comput. Sci.*, vol. 9, p. e1705, 2023.

[3] A. S. Yadav, N. Singh, and D. S. Kushwaha, “Evolution of blockchain and consensus mechanisms and its real-world applications,” *Multimedia Tools Appl.*, vol. 82, no. 22, pp. 34363–34408, 2023.

[4] S. N. Khan *et al.*, “Blockchain smart contracts: Applications, challenges, and future trends,” *Peer-to-Peer Netw. Appl.*, vol. 14, no. 5, pp. 2901–2925, 2021.

[5] N. Dimitrijević and N. Zdravković, “A review on security vulnerabilities of smart contracts written in Solidity,” in *Proc. Inf. Soc. Serbia (ISOS)*, Kopaonik, Serbia, 2024, pp. 10–13.

[6] E. Daspe *et al.*, “Benchmarking large language models for Ethereum smart contract development,” in *Proc. 6th Conf. Blockchain Res. Appl. Innov. Netw. Serv. (BRAINS)*, 2024, pp. 1–4.

[7] F. A. Alaba *et al.*, “Smart contracts security application and challenges: A review,” *Cloud Comput. Data Sci.*, pp. 15–41, 2024.

[8] M. Soud, “Advancing smart contract security: Vulnerability characterization, classification, and automated detection,” 2024.

[9] N. Sharma and B. Verma, “Recent advances in transfer learning for natural language processing (NLP),” in *A Handbook of Computational Linguistics: Artificial Intelligence in Natural Language Processing*, 2024, pp. 228–254.

[10] L. Zhang *et al.*, “A novel smart contract vulnerability detection method based on information graph and ensemble learning,” *Sensors*, vol. 22, no. 9, p. 3581, 2022.

[11] L. Heimbach *et al.*, “Deanonymizing Ethereum validators: The P2P network has a privacy issue,” in *Proc. 34th USENIX Secur. Symp.*, 2025, pp. 1319–1338.

[12] H. Xiong *et al.*, “Research on progress of blockchain consensus algorithm,” *Future Internet*, vol. 14, no. 2, p. 47, 2022.

[13] L. Brent *et al.*, "Ethainter: A smart contract security analyzer for composite vulnerabilities," in *Proc. ACM SIGPLAN Conf. Program. Lang. Des. Implement.*, 2020, pp. 454–469.
[14] N. Lu *et al.*, "NeuCheck: A more practical Ethereum smart contract security analysis tool," *Softw. Pract. Exp.*, vol. 51, no. 10, pp. 2065–2084, 2021.
[15] A. Ali, Z. U. Abideen, and K. Ullah, "SESCon: Secure Ethereum smart contracts by vulnerable patterns' detection," *Secur. Commun. Netw.*, vol. 2021, p. 2897565, 2021.
[16] N. F. Samreen and M. H. Alalfi, "SmartScan: An approach to detect denial of service vulnerability in Ethereum smart contracts," in *Proc. IEEE/ACM Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, 2021, pp. 17–26.
[17] Z. Yang, H. Lei, and W. Qian, "A hybrid formal verification system in Coq for ensuring the reliability and security of Ethereum-based service smart contracts," *IEEE Access*, vol. 8, pp. 21411–21436, 2020.
[18] J. F. Ferreira *et al.*, "SmartBugs: A framework to analyze Solidity smart contracts," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2020, pp. 1349–1352.
[19] G. A. Pierro, "Smart-graph: Graphical representations for smart contract on the Ethereum blockchain," in *Proc. IEEE Int. Conf. Softw. Anal. Evol. Reeng. (SANER)*, 2021, pp. 708–714.
[20] Y. Liu *et al.*, "ModCon: A model-based testing platform for smart contracts," in *Proc. ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 1601–1605.
[21] M. Ashouri, "Etherolic: A practical security analyzer for smart contracts," in *Proc. ACM Symp. Appl. Comput.*, 2020, pp. 353–356.
[22] T. D. Nguyen *et al.*, "sFuzz: An efficient adaptive fuzzer for Solidity smart contracts," in *Proc. ACM/IEEE Int. Conf. Softw. Eng.*, 2020, pp. 778–788.
[23] T. Chen *et al.*, "SODA: A generic online detection framework for smart contracts," in *NDSS*, 2020.
[24] Y. Zhuang *et al.*, "Smart contract vulnerability detection using graph neural networks," in *Proc. Int. Joint Conf. Artif. Intell.*, 2021, pp. 3283–3290.
[25] Z. Zhen *et al.*, "DA-GNN: A smart contract vulnerability detection method based on dual attention graph neural network," *Comput. Netw.*, vol. 242, p. 110238, 2024.
[26] S.-J. Hwang *et al.*, "CodeNet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection," *IEEE Access*, vol. 10, pp. 32595–32607, 2022.
[27] W. Wang *et al.*, "Blockchain-enabled authentication handover with efficient privacy protection in SDN-based 5G networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1133–1144, 2021.
[28] N. Ashizawa *et al.*, "Eth2Vec: Learning contract-wide code representations for vulnerability detection on Ethereum smart contracts," in *Proc. ACM Int. Symp. Blockchain Secure Crit. Infrastruct.*, 2021, pp. 47–59.
[29] J. Huang *et al.*, "Smart contract vulnerability detection model based on multi-task learning," *Sensors*, vol. 22, no. 5, p. 1829, 2022.
[30] C. Xing *et al.*, "A new scheme of vulnerability analysis in smart contract with machine learning," *Wireless Netw.*, vol. 30, no. 7, pp. 6325–6334, 2024.
[31] L. S. H. Colin *et al.*, "An integrated smart contract vulnerability detection tool using multi-layer perceptron on real-time Solidity smart contracts," *IEEE Access*, vol. 12, pp. 23549–23567, 2024.
[32] L. Zhang *et al.*, "A novel smart contract reentrancy vulnerability detection model based on BiGAS," *J. Signal Process. Syst.*, vol. 96, no. 3, pp. 215–237, 2024.
[33] H. Wu *et al.*, "Smart contract vulnerability detection based on hybrid attention mechanism model," *Appl. Sci.*, vol. 13, no. 2, p. 770, 2023.
[34] Y. Sun *et al.*, "GPTScan: Detecting logic vulnerabilities in smart contracts by combining GPT with program analysis," in *Proc. IEEE/ACM Int. Conf. Softw. Eng.*, 2024, pp. 1–13.
[35] I.-F. Su *et al.*, "A smart contract vulnerability detection manner based on large language model," in *Int. Conf. Knowl. Innov. Invent.*, 2024, pp. 16–25.
[36] S. Chopra *et al.*, "RoBERTa and BERT: Revolutionizing mental healthcare through natural language," *SN Comput. Sci.*, vol. 5, no. 7, p. 889, 2024.
[37] S. B. Ahmed *et al.*, "Text in the wild and its challenges," in *Cursive Script Text Recognition in Natural Scene Images: Arabic Text Complexities*. Springer, 2019, pp. 13–30.

[38] A. Kaladevi *et al.*, "Tokenization and its applications," in *Human-Centric Integration of Next-Generation Data Science and Blockchain Technology*. Elsevier, 2025, pp. 147–164.
[39] S. Vijayarani and R. Janani, "Text mining: Open source tokenization tools—An analysis," *Adv. Comput. Intell. Int. J.*, vol. 3, no. 1, pp. 37–47, 2016.
[40] L. Stankevičius and M. Lukoševičius, "Extracting sentence embeddings from pretrained transformer models," *Appl. Sci.*, vol. 14, no. 19, p. 8887, 2024.
[41] W. Rahayu *et al.*, "Synthetic minority oversampling technique (SMOTE) for boosting the accuracy of C4.5 algorithm model," *J. Artif. Intell. Eng. Appl.*, vol. 3, no. 3, pp. 624–630, 2024.
[42] J. Y. Lim *et al.*, "SSL-ProtoNet: Self-supervised learning prototypical networks for few-shot learning," *Expert Syst. Appl.*, vol. 238, p. 122173, 2024.
[43] A. Rana *et al.*, "A computerized analysis with machine learning techniques for the diagnosis of Parkinson's disease," *Diagnostics*, vol. 12, no. 11, p. 2708, 2022.
[44] Y. Song *et al.*, "A comprehensive survey of few-shot learning," *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–40, 2023.
[45] H. Gharoun *et al.*, "Meta-learning approaches for few-shot learning," *ACM Comput. Surv.*, vol. 56, no. 12, pp. 1–41, 2024.
[46] J. M. Valls *et al.*, "Supervised data transformation and dimensionality reduction with a 3-layer MLP," *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 12, pp. 10515–10527, 2021.
[47] D. K. Mandarapu *et al.*, "Arkade: k-nearest neighbor search with non-Euclidean distances using GPU ray tracing," in *Proc. ACM Int. Conf. Supercomput.*, 2024, pp. 14–25.
[48] A. Mao, M. Mohri, and Y. Zhong, "Cross-entropy loss functions: Theoretical analysis and applications," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 23803–23828.
[49] R. K. Halder *et al.*, "Enhancing k-nearest neighbor algorithm," *J. Big Data*, vol. 11, no. 1, p. 113, 2024.
[50] H. A. Salman *et al.*, "Random Forest algorithm overview," *Babylonian J. Mach. Learn.*, pp. 69–79, 2024.
[51] M. Aoun and R. Holmdahl, "Self-antigens select B cells," *Eur. J. Immunol.*, vol. 55, no. 7, p. e51720, 2025.
[52] M. Schonlau, "Applied statistical learning," in *Statistics and Computing*. Springer, 2023, pp. 143–160.