Block-Chain Security Enhancement: Using Deep Learning to Detect Smart Contract Vulnerabilities

Prof. Prachi A. Bainalwar[1], Sayuri Wankhede[2], Snehit Telrandhe[3], Pratham Channujwar[4], Pratik Hekad[5], Lavannya Rangari[6], Vedant Kshirsagar[7]
[1,2,3,4,5,6,7] *Department of Computer Technology Yeshwantrao Chavan College of Engineering, Nagpur, India*

***Abstract*—Blockchain technology has revolutionized the digital ecosystems by introducing a decentralized, transparent, and tamper-resistant systems that can eliminate the need for intermediaries among its various implementations, Ethereum is one of the leading platforms supporting the smart contracts which are the self-executing program that can automatically enforce the agreements on the blockchain. However, the rapid adoption of smart contracts has also exposed them to serious security threats. One of the most serious vulnerabilities is reentrancy. Reentrancy remains one of the most critical and exploited vulnerabilities in Ethereum smart contracts, enabling attackers to repeatedly invoke external calls before state updates and drain funds, as seen in DAO attack. Traditional detection methods- such as static analysis, symbolic execution and fuzz technique- are limited by expert-defined rules and struggle to uncover novel or evolving attack patterns. To address these shortcomings, this project proposes an automated deep-learning framework dedicated to reentrancy vulnerability detection. Solidity Smart Contracts are converted into the Abstract Syntax Tree (AST) and a graph-based representation. This process helps to capture all the present syntactic and semantic dependencies. After all these the structured features are then embedded into the neural network models, that is the Recurrent neural networks (RNNs) and another one is the Graph neural networks (GNNs), for a good precise vulnerability classification and the detection. The proposed approach combines both the static and the dynamic analysis which helps to improve the recall and also low-down the false positives which enables the robust detection of all the complex reentrancy patterns. At the end a proper user-friendly API and a web interface is being developed which provides the real-time contract scanning and detailed security reports. By focusing on the reentrancy attack detection, the system that has been proposed offers a scalable, accurate, and an automated security auditing that tries to significantly strengthen the reliability of the Ethereum-based decentralized applications.**

***Index Terms*—Block-Chain, Ethereum, Smart Contacts, Deep Learning, Vulnerability Detection, Solidity Code**

I. INTRODUCTION

Blockchain technology has become a gamechanger for the secure and decentralized digital transactions as it removes a need for trusted intermediaries [1][2][5]. Among all of its most important innovations are the Smart Contracts, which are the self-executing program that are being deployed on blockchain that automatically enforces the terms of an agreement, that runs on a platform like Ethereum. The most unchangeable nature of the Smart Contracts ensures the transparency and the trust, but it also brings a significant security issue [6]. Once it is being deployed, the code cannot be changed, making any of the hidden flaws permanent and easy to exploit. Notable incidents that occurred previously are the Decentralized Autonomous Organization (DAO) hack [4] and the Binance breach have shown that subtle weaknesses, especially the reentrancy attacks, can lead to a huge financial loss and undermine a reliability of the decentralized applications (dApps).

Reentrancy remains one of the most severe and very frequently exploited vulnerabilities in the Ethereum smart contracts [26][28]. It allows a malicious contract to repeatedly call back into a vulnerable contract before its internal state is updated, that enable attackers to drain funds or manipulate the execution flow [31][33]. Existing detection approaches- that includes static analysis, symbolic execution and fuzz testing [12] primarily rely on the expert-defined rules and heuristics. Although effective for known patterns, these patterns often fail to detect the zero-day attacks and a novel reentrancy variant arising from the evolving nature of Solidity and smart contract design.

This necessitates the development on an intelligent, adaptive, and an automated vulnerability detection framework that can effectively address the limitations of the traditional techniques.

Deep Learning offers a powerful data-driven solutions by automatically recognizing the hidden patterns and the complex relationships in the smart contract code. The proposed framework includes a deep learning-based system which focuses on detecting the reentrancy vulnerabilities in the Ethereum smart contracts. This approach converts Solidity code into an Abstract syntax tree (AST) [4][26] and a graph representation. This conversion captures the code's syntactic and semantic dependencies. The structured representations are then embedded and processed using the neural architectures like Recurrent neural networks (RNNs) and another one is Graph neural networks (GNNs). This method supports an accurate and a scalable vulnerability classification. Also, combining static and dynamic analysis techniques boosts detection accuracy and lowers false positives. This ensures a strong identification of complex reentrancy patterns.

RNNs primarily support the sequential processing of data. This specific kind of neural network shows the dependencies of the elements in the sequences through the hidden states which are capable of keeping the context from the previous steps. In the case of smart contract analysis, RNNs are able to model the sequential patterns that are related to the execution of the code and thus are able to determine if there are any weaknesses in smart contracts based on the called functions or the transactions flow [7][9]. They are great for comprehending time and spotting reentrancy that could happen during the execution of a smart contract.

Conversely, Graph neural networks (GNN) [30] are good at extracting information from data that is in the form of graphs. They look at the nodes (indicative of the program parts) and the edges (indicating relations like data flow or control flow) to be able to observe both the grammatical and the meaning connections of the contract code. Graph Neural Networks are remarkably efficient not only in recognizing complicated structural interrelations among functions and variables [11], but also in making the model conscious of various contract sections and the routes through which advantage of vulnerabilities can be taken. The most fundamental difference between the two models lies in the type of data they deal with and the connections they hypothesize. Like RNN, GNN [25] has an immediate access to the interconnections among the nodes [13]. So, both models obtain the input this way but in different manners [17][19] i.e. RNN shows the smart contracts execution paths while GNN shows the architectural structure of the same.

The key contributions of the proposed approach are as follows:

- Improving the pre-processing of code and the feature representation: The system parses the Smart Contracts into an Abstract syntax tree (AST) [4] and a graph-based data structures[30]. These methods most effectively captures both the control-flow and the data-flow semantics.
- Detection with the use of Deep Learning Models: The Recurrent neural networks (RNNs) and the Graph neural networks (GNNs) [25] model analyzes all the high-dimensional feature embeddings to perform an accurate classification of the reentrancy vulnerabilities.
- Hybrid which includes static and dynamic analysis: The method combines both static code analysis with dynamic execution monitoring. The method enables users to find the security weaknesses, which remains hidden when examining the original source code.
- An Automator and a Scalable Security Auditing: A RESTful API and an interactive web interface that provides the real-time contract scanning and a detailed vulnerability reports to the developers and the auditors.
- High Reliability and High Flexibility: The system is built to adjust all the changing Solidity patterns. It can also detect zero-day reentrancy exploits accurately while keeping the precision high and the false rates low.

The motivation behind this is using the power of deep learning through RNNs and GNNs into an effective and scalable manner to build a framework capable of detecting reentrancy attacks [34] in Ethereum smart contracts. Combining deep learning with static and

dynamic analysis, this system will strengthen the auditing in security, reduce human intervention, and enhance trust in blockchain ecosystems. By focusing on detecting the reentrancy vulnerabilities, this research provides an automated, a scalable, and an accurate security auditing system that greatly improves the trust and the reliability of the Ethereum-based decentralized applications (dApps), making them more secure and resilient in the real-world use. The primary intention of the current dissertation is to build an automated hybrid system based on deep learning techniques that will be used for detecting reentrancy vulnerabilities in smart contracts on the Ethereum platform. In doing so, it indirectly contributes to strengthening the overall security of blockchain technology. The analysis of restrictions related to existing detection methods, which include static analysis, symbolic execution, and fuzz testing, will be the central point of the study. These methods, in most cases, are unable to detect the newly emerging or unrecognized attack patterns [22]. Thus, the doctorate works on establishing a solid framework consisting of Recurrent neural network (RNN) and the other one is the Graph neural network (GNN) components for the precise bifurcation of vulnerable and non- vulnerable contracts.

Moreover, an emphasis is put on the effective preprocessing and representation of features where Solidity smart contracts are converted into Abstract Syntax Trees (ASTs) [38] besides graph- based structures to represent not only the syntactic but also the semantic relations. The amalgamation of static and dynamic analysis is expected to lead the system to a point where it possesses high detection accuracy and low false positives. Even more, a RESTful API along with a web-based interface are developed as tools for the purpose of being able to conduct real-time scanning and reporting of contracts. The proposed models are to be appraised by employing accuracy, precision, recall, and F1-score metrics that are considered standard in the evaluation of performance, thus ensuring the reliability and scalability of the models in the real-world application.

II. LITERATURE SURVEY

The detection of vulnerabilities in the smart contract has been developed in recent years to utilize the machine learning and the hybrid analysis techniques to go beyond the limitations of all the traditional static tools. For an example, SCSVM and SCLMF [1] leverage the support vector machines and meta-learning, respectively, to enhance the classification performance and mitigate issues of mostly small datasets with reliable accuracy in all the tested conditions. Alternatively, there are some more approaches that focus on the front-running vulnerabilities through attack mining; however, while this may also improve recall, this tool cannot overcome the limitations in detecting the completeness [2]. The study proposed by Q. Zhou et al. focuses on the tree-based (neural) models, such as TMLVD, uses an Abstract syntax tree (AST) [3] to detect the vulnerabilities with a high accuracy and a sub-second speed, although they have the same limitations in vulnerability localization and scope knowledge. Similarly, multi-layer perceptron one of the (MLP)-based tools with the Control Flow Graph (CFG) and opcode features also support the high accuracy with the low false positives aided by a balanced training dataset created with a standardized preprocessing & bug injection [4], and like TMLVD, they are still limited in contextual understanding and localization.

The study by Z. Liu et al. identified the hybrid systems such as in combining the Graph neural networks (GNNs) with an expert defined features achieve a focus vulnerability detection like reentrancy and timestamp dependence, perform well on the Ethereum/VNT datasets [5], but lack a bytecode-level analysis and a broader detection scope. Moreover, the Transformer-based models in detecting the smart contracts fine-tuned to detection perform well with the ROC-AUC scores [6] close to 1.0 and also increase performance on minority classes through oversampling, but struggle with a multi-label classification are relatively costly in terms of computing resources. More such complex architectures such as the VMD-AEI, achieved high F1-scores through a greyscale image- based feature extracting and a deep dual-interaction networks, but the models required much computation [7]. Similarly, the study evaluated by Y. Cao et al. introduces SCcheck, a GNN-based framework, modeled on the bytecode level requiring a multi-head attention to processes examines, performs better than the models older than the LSTM and the CNN-GRU based models

[8], but is still limited to the static analysis with the possibility of expanding it to dynamic execution.

Novel methodologies such as the Semantic Contract Graph (SCG) approach obtain a high precision and a recall by graph- matching against all the known patterns, but they are hindered by the issues such as lack of standardized benchmarks and being leak-prone against adversarial evasion [9]. According to S. S. Kushwaha et al. the systematic reviews continue to highlight the deficiencies of common tools, such as Mythril, Oyente, and Slither [10][11], with narrow detection scopes and even more problems, stressing the need for an AI-driven and a hybrid framework to combat various tolerable vulnerabilities such as the reentrancy or timestamp blockchain vulnerabilities. While broad evaluations conducted on 27 different tools spanning categories of formal verification to deep learning acknowledge formal methods have the broadest coverage for all the vulnerabilities, deep learning models have the highest accuracy coupled with false positives, and patchy backward compatibility for the outdated contracts [12]. Direct studies have also highlighted the vulnerabilities like an insecure randomness which indicate that over 80% of contracts employed undue Random Number Generation (RNG) [13] leading to multimillion-dollar consequences; tools like the RNVulDet can delineate points of vulnerabilities with a good accuracy and a good speed but call for an increased need to improve the randomness mechanisms.

The latest hybrid detection approaches, such as those that integrate the Graph neural networks (GNNs) with an EVM-level expert defined rules, utilizes a two-phase architecture, improving a runtime protection and accuracy, all with a reasonable performance impact on the system [14]. Similarly, Z. Liu et al. in their study they introduced new models like the CodeNet re-purpose bytecode into the RGB images for an improved semantic interpretation, allowing for rapid, and accurate detection, yet it also require vast amounts of labelled datasets and confront false positives regardless of the type of the vulnerability [15]. The various methods and their development demonstrate a clear pattern that the smart contract security will evolve into a hybrid system which combines the AI-based techniques with a specialized domain knowledge, deep learning capabilities, and thorough benchmarking methods.

Blockchain-based decentralized applications (DApps) have become an extremely popular because they provide transparent, unchangeable, and distributed computation functions. The smart contract vulnerabilities, especially reentrancy attacks, create a major danger to blockchain ecosystem security. The traditional static, symbolic, and fuzz-based detection methods often struggle to capture the dynamically evolving attack patterns, thereby necessitating the adoption of the deep learning-based intelligent security models. All the recent research has demonstrated the potential of the neural architectures in identifying the complex and the non-linear dependency structures within data. For an instance, Pande et al. [16] introduced the HybridLSTM model that utilizes a hybrid feature extraction and a long-short-term memory networks for an adaptive pattern recognition, a concept extendable to identifying the dynamic transactional dependencies in the Ethereum contracts. Similarly, Khekare et al. [17] explored an artificial intelligence algorithm for decision optimization, laying a foundation for applying a deep learning mechanisms to autonomous vulnerability classification and blockchain auditing.

In the world of the cybersecurity, Nyangaresi and Yenurkar [18] proposed a lightweight authentication protocol that is being designed for a resource-constrained networks, used for addressing a distributed security challenges relevant to the blockchain's decentralized design. Moreover, Nyangaresi et al. [19] presented an unnamed authentication framework based on the physically unique functions and biometrics, underscoring the role of an anonymity and a cryptographic integrity in enhancing the blockchain-based security mechanisms. In addition, Assudani et al. [20] emphasized on the application of chaos engineering principles that evaluate the resiliency in the cloud services, which aligns conceptually with the robustness testing of the smart contracts under an adversarial and a high-load scenarios. Complementing the same work, Yenurkar and Mal [21] conducted a comprehensive review of a big data and the machine learning algorithms, offering a crucial insight into the performance optimization and a reliability of the neural network models in the security-oriented tasks. Collectively, all the studies establish a foundation for employing the hybrid (static and dynamic) deep learning architectures, such as the Recurrent neural networks (RNNs) and the Graph neural networks (GNNs) frameworks, for an

automated reentrancy vulnerability detection and a scalable blockchain security enhancement.

III. METHODOLOGY

EXISTING METHODOLOGY

Blockchain technology creates the foundation of secure and transparent digital ecosystems, where the smart contracts automate the trust less transactions [1]. However, the most common vulnerabilities such as reentrancy, timestamp dependence, and integer overflow compromise the reliability of the decentralized applications (dApps).

1. Static Analysis Tool (Mythril, Oyente, Slither) [10][11]

Static analysis tools inspect the smart contract code without any execution, identifying the syntactic or logical flaws before doing deploying. These are one of the earliest blockchain vulnerability detectors.

- How it works: It analyse the solidity source code or bytecode using an Abstract Syntax Tree (AST) and a Control Flow Graph (CFG) parsing.
- Purpose: It helps to detect the known vulnerability issues i.e. the reentrancy attack, integer overflow, and the timestamp dependence.
- Advantages: Fast, scalable, and works pre-deployment.
- Limitations: It is very difficult to capture the runtime or the dynamic vulnerabilities.

2. Symbolic Execution and Fuzz Testing [10][11]

The dynamic techniques explore the behaviour of the contract during an execution using the symbolic inputs or random testing data. They are important for runtime vulnerability detection.

- How it works: The symbolic execution mostly explores all the execution paths symbolically, while the fuzz testing uses the random inputs to trigger certain anomalies.
- Purpose: Helps in identifying the behavioural and the hidden logical flaws missed by the static methods.
- Advantages: It helps in detecting the real execution-level errors.
- Limitations: Computational cost is very high and it also shows path explosion issues.

3. Machine Learning Models (SCSVM, SCLMF) [3][14]

Machine Learning introduces a data-driven detection by classifying the smart contracts based on some learned patterns which helps in reducing the manual rule-dependence.

- How it works: It trains the SVM and meta-learning models on the labelled datasets which mainly contains vulnerable and smart contracts data.
- Purpose: Automates the detection of all known and unknown vulnerability types.
- Advantages: Handles small data set with a good accuracy.
- Limitations: Has limited contextual understanding and interpretability.

4. Transform-Based Models [6][8]

Transform models holds the attention on the mechanisms to identify the long-range dependencies in the smart contract code.

- How it works: It works by tokenizing the code and then it encodes the code sequences by applying the multi-head attention for the contextual learning.
- Purpose: Enhances detection accuracy for complex vulnerabilities.
- Advantages: Has high ROC-AUC and better minority class handling.
- Limitations: It is computationally expensive and hard to train.

5. Image-Based Feature Extraction (CodeNet, VMD-AEI) [15][14]

They transform the smart contract bytecode into a visual data and also identifies the patterns using deep learning.

- How it works: It converts the EVM bytecode into the RGB or grayscale image for CNN-based feature extraction.
- Purpose: It helps to capture all the hidden relationships in the smart contract semantics.
- Advantages: It detects high precision and innovation feature representation.
- Limitations: Mostly requires a large labelled datasets and a high computing power.

6. RNVulDet for Randomness Vulnerabilities [13]

RNVulDet model targets the smart contracts with an insecure random number generation mechanism.

- How it works: It work by analysing the bytecode for weak or predictable random functions.
- Purpose: It helps in detecting the RGN flaws which further leads to exploit the risks.
- Advantages: The model is specialized and precise for analysing the randomness.
- Limitations: Its one of the limitations includes that it has the limited detection scope beyond the RNG issues.

7. Anonymous Authentication and **Cryptographic** Mechanisms [19][18]

On the blockchain-level cryptographic mechanism, this model enhances the transaction and contract authentication security.

- How it works: The model uses a Physical Unclonable Functions (PUFs) and biometrics for verification of identity.
- Purpose: It plays a good in preventing the unauthorized access and contract tampering.
- Advantages: The model helps to strengthens the privacy and most of the identity security.
- Limitations: Mainly increases the system complexity and cost

Table 1. Existing Methodologies with main idea, advantages, and its limitations

Detection Method	Name	Main ideas	Advantages	Limitations
Symbolic Execution [10][11]	Mythril	It designd a special symbolic execution library based on the EVM-compatible bytecode for the purpose of the Concolic analysis, taint analysis and the control flow checking.	1. Low detection error rate. 2. Detect multiple vulnerabilities simultaneously. 3. It tries to report how the vulnerability is being triggered.	1. Long detection time. 2. High computational cost. 3. It requires a pre-define symbolic patterns. 4. It tries to Influence the detection coverage due to the symbolic constraints. 5. It is unable to detect the unknown vulnerabilities.
	MythX	It expands the Mythril and be compatible with the multiple EVM-based platforms.		
	Oyente	It tries to extract the information from the Control flow graph (CFG) and the current Ethereum global state.		
	Manticore	It generates the symbolic transactions to explore the state space of the smart contract.		
	Securify	It analyses the smart contract's dependency graph and also checks the compliance and violation patterns		
	SmartCheck	It performs the static analysis using an XML-based intermediate representation and XPath patterns.		
Formal Verification [10]	Remix	Perform static analysis without actually executing the code.	1. Medium detection time. 2. Medium computational cost. 3. Efficiently detect	1. It shows low scalability. 2. It gives the high detection error rate.
	Slither	It performs the Static single assessment (SSA) based on the		

		designed intermediate representation SlithIR.	common simple vulnerabilities.	3. It requires a pre-construction of the abstract formal models. 4. It is unable to detect the unknown vulnerabilities.
	ZEUS	It translates the smart contracts and the policy specification into an abstract interpretation, and uses the constrained horn clauses to verify the models.		
Fuzzing [10][11]	ContractFuzzer	It generates the fuzzing inputs based on the bytecode and then defines the test oracles to detect the vulnerabilities.	1. Detect multiple vulnerabilities simultaneously.	1. High detection error rate. 2. It require pre-define detection oracles.
Machine Learning [3][10]	SoliAudit	It designs a vulnerability analyser based on the opcode sequence and a gray-box fuzz testing mechanism.	1. It shows low detection time. 2. There is no need to pre-define the detection model	1. Require labeled dataset. 2. Long training time.
Transform-Based Models [6]	Transformer-based Detectors	Leverage attention mechanisms to identify long-range dependencies in smart contract code. Tokenizes and encodes code sequences, applying multi-head attention for contextual learning.	1. High ROC-AUC. 2. Better handling of minority classes. 3. Enhanced accuracy for complex vulnerabilities.	1. Computationally expensive. 2. Hard to train.
Image-Based Feature Extraction [15]	CodeNet, VMD-AEI	Convert EVM bytecode into RGB or grayscale images for CNN-based feature extraction.	1. High detection precision. 2. Innovative feature representation. 3. Captures hidden semantic relationships.	1. Requires large labeled datasets. 2. High computing power.
Randomness Vulnerability Detection [13]	RNVulDet	Analyses bytecode for weak or predictable random functions to detect insecure random number generation mechanisms.	1. Specialized and precise for randomness analysis.	1. Limited detection scope beyond RNG issues.
Anonymous Authentication and Cryptographic Mechanisms [19]	-	Uses Physical Unclonable Functions (PUFs) and biometrics for identity verification to prevent unauthorized access and contract tampering.	1. Strengthens privacy and identity security.	1. Increases system complexity and cost.

PROPOSED METHODOLOGY

The approach introduces a deep learning framework for automatically detecting the vulnerabilities in the smart contracts. The objective for this is to enhance the

security of the blockchain applications. This system scans a source code and then considers the 2 types of the neural network architectures out of which one is the Recurrent neural networks (RNNs) and another one is the Graph neural networks (GNNs) [25][30]. It identifies the smart contracts as a non-vulnerable i.e. safe or a vulnerable i.e. not safe. The goal here is to evaluate and compare these two architectures based on their ability to capture code meaning, the structural relationships, and the patterns that may indicate potential security issues.

This process begins with the PrimeSmartVuln dataset, containing the collection of Solidity smart contracts, in which they are all being labeled to indicate whether it is vulnerable, unsafe, or not vulnerable, safe. The raw contract code is passed through a preprocessing pipeline, consisting of tokenization, normalization, and sequence formatting, which make it uniform and compatible with the neural network models [19]. In the case of the RNN-based models, the code in the smart contract is being usually transformed into the sequential form, where the code is expressed as the sequence of tokens or the numerical embeddings such that the order and the context of the instructions are preserved. This enables the RNN model to learn the temporal and the contextual patterns in the code so that the model can learn how the sequences of statements may imply the possible vulnerabilities. For GNN-based models, the code is being parsed further into a structured form such as the Abstract Syntax Trees (ASTs) that retain the hierarchical structure and the execution flow of the contracts.

Once preprocessing is complete, the smart contracts are then subjected into a feature extraction for each model. For the RNN model, the token sequences are then transformed into a dense vector embedding and then processed step by step, allowing the model to capture the dependencies and the patterns between code statements. On the other hand, the GNN model works directly on the graph representation [31][38] of the contract, that spreads information across nodes to detect the structural patterns that may signal the vulnerabilities. Each of the models produces a prediction score through a dense layer with a sigmoid activation, which indicates the probability that a given smart contract is being vulnerable. Then a binary decision is been made based on the predefined threshold. The Adam optimizer is employed to adjust adaptively to the learning rate for more efficient and stable training, while the models are being trained using the binary cross-entropy loss function. To improve the generalization, dropout layers, early stopping, and learning rate, scheduling are incorporated during training, along with a validation, that is set to monitor and prevent overfitting. The actionable outcome of this system is a binary classification label for each of the contracts: vulnerable (not safe) or non-vulnerable (safe). The system is being designed to be extendable to the multi-label or multi-class settings in future work, where the specific types of vulnerabilities may also be predicted.

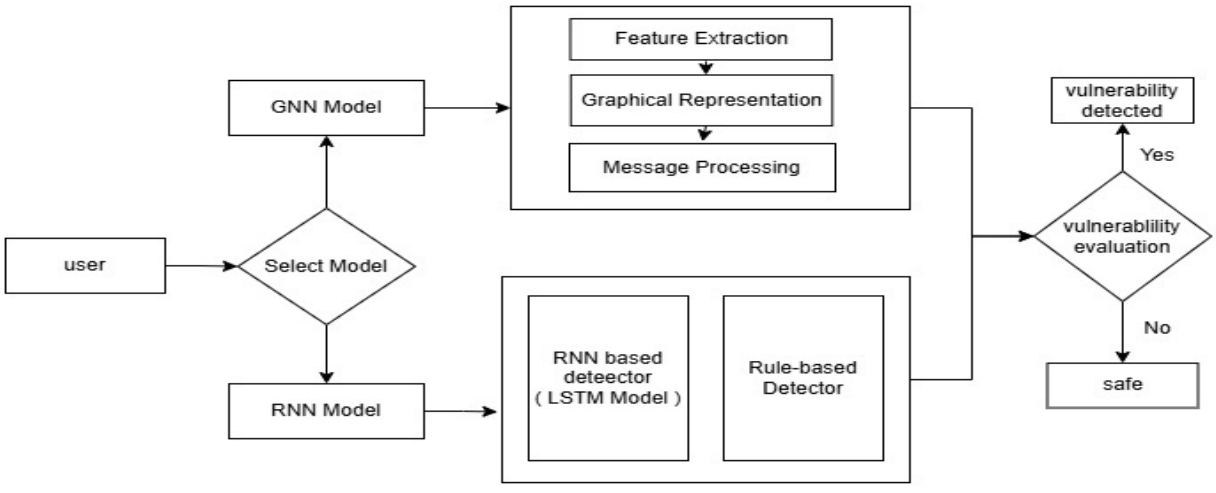

Figure 1. System Architecture Diagram

This system architecture diagram above represents a hybrid smart contract vulnerability detection framework that combines both the machine learning and the rule based approaches for the precise analysis. The process begins with the user inputting a smart contract, after which the system performs a model selection step to decide whether the use of the Graph neural network (GNN) [17] or the Recurrent neural network (RNN) model based on the type of the contract data and its structure. If the GNN model is being selected, it processes the contract through the feature extraction, graphical representation and the message passing which converts the contract code into a structured graph that captures the data and the control flow relationships.

In contrast, if the RNN model which includes specially the LSTM-based detector [4] is being selected, which then try to analyse the sequential patterns of the contract's operations to identify all the behavioural anomalies. Also, after this the rule based detector operates to cross-verify the results based on all the predefined vulnerability patterns and the static rules. The resulting outputs from all these components are then sent to the vulnerability evaluation module, which helps to determine whether any of the vulnerabilities are being detected or not. If the system finds any of the abnormal patterns or any ruled violations, it flags the contract as vulnerable; otherwise, it classifies it as the contract is safe. This hybrid approach helps in enhancing the detection accuracy by combining all the contextual understanding of the neural networks with the precision of the expert-defined rules.

IV. WORK DONE

1. Training Of Dataset

PrimeSmartVuln is the dataset used for this system, which contains a vast amount of labelled Solidity smart contracts. Each contract in the dataset is annotated as either vulnerable (unsafe) or non-vulnerable (safe) based on whether it contains one or more known vulnerabilities such as reentrancy attacks. The dataset is divided into three subsets to effectively train the deep learning models:

- 70% for training finds its usage where a model learns patterns or relationships present within the data.
- 15% for validation: This will be utilized to fine-tune the model parameters and avoid overfitting.
- 15% for testing: this will be used in evaluating the performance of the model on unseen data.

Both the RNN and GNN are trained by being exposed to labelled examples during training, so they can learn to identify patterns in smart contract code that indicate the presence of a potential vulnerability.

The process of training is the minimization of a binary cross-entropy loss that measures the closeness of the model's predictions to actual labels. Adaptive control of the learning rate for each parameter is done by using the Adam optimizer, which allows for more stable and efficient optimization.

Several techniques are also implemented to improve model performance:

- Added dropout layers to prevent overfitting by randomly disabling neurons during training.
- Early stopping: This is a form of regularization that stops training if the model's performance on the validation dataset starts degrading. The reasoning behind doing this is that it helps the model generalize optimally.
- Learning rate scheduling dynamically adjusts the learning rate to avoid sudden jumps or slow convergence.

This systematic training process allows them to learn complex representations of data and classify smart contracts as vulnerable or non-vulnerable with high accuracy.

2. Preprocessing

Before training the deep learning models, the raw Solidity smart contract code needs to be preprocessed into a representation suitable for machine learning. Preprocessing is a common practice aimed at cleaning and normalizing data in a manner that gets them ready for model input. The preprocessing pipeline consists of several systematic steps:

2.1 Tokenization:

The source code of the smart contract is divided into small, meaningful elements called tokens. These tokens are keywords, identifiers, operators, numbers, and symbols used in a program. Tokenization helps convert the unstructured source code into a structured sequence that can be processed by neural networks.

2.2 Normalization:

After tokenization, normalization is applied to remove

unnecessary symbols, comments, and formatting inconsistencies. It also normalizes all code into a standardized form, which can include things like consistent spacing and all lowercase to make sure that the model doesn't treat different writing styles as different patterns.

2.3 Sequence Formatting:
Since smart contracts have different lengths, the tokenized sequences are either padded, in case of shorter sequences, or truncated if they are longer than the desired input size to maintain a uniform input size across all samples. This uniformity is of utmost importance for RNN based models in performing batch processing with ease.

2.4 Graph Construction - GNN Models:
Besides linear sequences of tokens, code is most naturally parsed into graph-based structures such as Abstract Syntax Trees or Control Flow Graphs. These structures capture the hierarchical and semantic relationships among the various components of the contract, such as function calls, variable dependencies, and control statements.
This step allows the GNN model to analyse not only the code sequence but also how different parts of code logically interact. Following the mentioned steps for pre-processing, the dataset is transformed into well-organized numerical or graphical formats that preserve syntactic and semantic information in code. This ensures that the RNN and GNN models receive consistent, meaningful inputs, leading to more accurate and efficient vulnerability detection.

3. Feature Extraction of Models
Feature extraction is the subsequent important step after preprocessing, whereby the structured smart contract data is transformed into numerical representations that the deep learning models can interpret and learn from. Since the project has two different architectures, namely the Recurrent Neural Networks and Graph Neural Networks, the feature extraction process varies slightly for each model.

3. 1 Feature Extraction for RNN Model
In the RNN model, the input data used is the tokenized sequences from Solidity code. Each token, which is either a keyword, a name of a variable, or an operator, among others, is mapped into a numerical vector using an embedding layer. This is called word embedding, where tokens that convey similar meanings or are used in similar contexts will have similar vector representations.

- These embeddings help the model capture semantic relationships in the code, such as recognizing that transfer and send are essentially the same function.
- RNN takes these vectors and processes them in sequence, maintaining the temporal order of code statements.
- It learns, through numerous iterations, the patterns, dependencies, and contextual relationships that may give signs of possible vulnerabilities; for example, repeated calls before a state update a characteristic of reentrancy attacks.

3.2 Feature Extraction for GNN Model
For the GNN model, the smart contract code is first converted into graph structures such as Abstract Syntax Trees or Control Flow Graphs.
In these graphs:

- Each node represents a code element (e.g., function, variable, or statement).
- Each edge represents a relationship or dependency between nodes, such as data flow, function call, or control sequence.
- For each node, a feature vector is assigned considering its attributes and its role in the code.

The GNN uses a message-passing mechanism where nodes exchange information with their connected neighbours. In this way, the network learns the structural and contextual dependencies necessary to spot complex code interactions that can lead to vulnerabilities.

3.3 Purpose of Feature Extraction
These features help the models in:

- Syntax, or the way the code is written and structured.
- Semantic understanding: This refers to the true meaning of the code—that is, what it really does.
- Identifying hidden patterns that may indicate vulnerable behaviours, even if those are not directly defined by any rule.

Thus, the effective feature extraction empowers the RNN and GNN models to transform raw Solidity code

into meaningful representations at high dimensions that can be analysed to provide an accurate and reliable vulnerability detection.

4. Model Architecture and Development

The proposed system is aimed at classifying Ethereum smart contracts into vulnerable and non-vulnerable categories using two deep learning architectures: Recurrent Neural Network and Graph Neural Network. Each of the models is crafted to capture different code structure and behavioural features, ensuring that detection is comprehensive.

4.1 Recurrent Neural Network Architecture

The RNN model is well-suited for learning from sequential data. In this context, smart contract code is represented as a sequence of tokens obtained from preprocessing.

The architecture consists of the following key components:

- Embedding Layer: Embeds tokens into dense vector representations called embeddings, which might capture some contextual relationships of code elements.
- Recurrent Layers (LSTM/GRU): These layers process the sequence of embeddings step by step while preserving the dependency between preceding and succeeding tokens. Long Short-Term Memory (LSTM) units are one of the most common methods to combat vanishing gradients and preserve long-term contextual information.
- Fully Connected Dense Layer: This receives the last output from the recurrent layer and maps it onto a high-level feature representation.
- Output Layer: The output from the sigmoid activation function gives a probability score, reflecting the likelihood that a contract is vulnerable.

The proposed RNN architecture learns temporal dependencies and patterns in smart contract execution flows effectively, which is important for identifying those vulnerabilities such as reentrancy that are dependent on the sequencing of function calls.

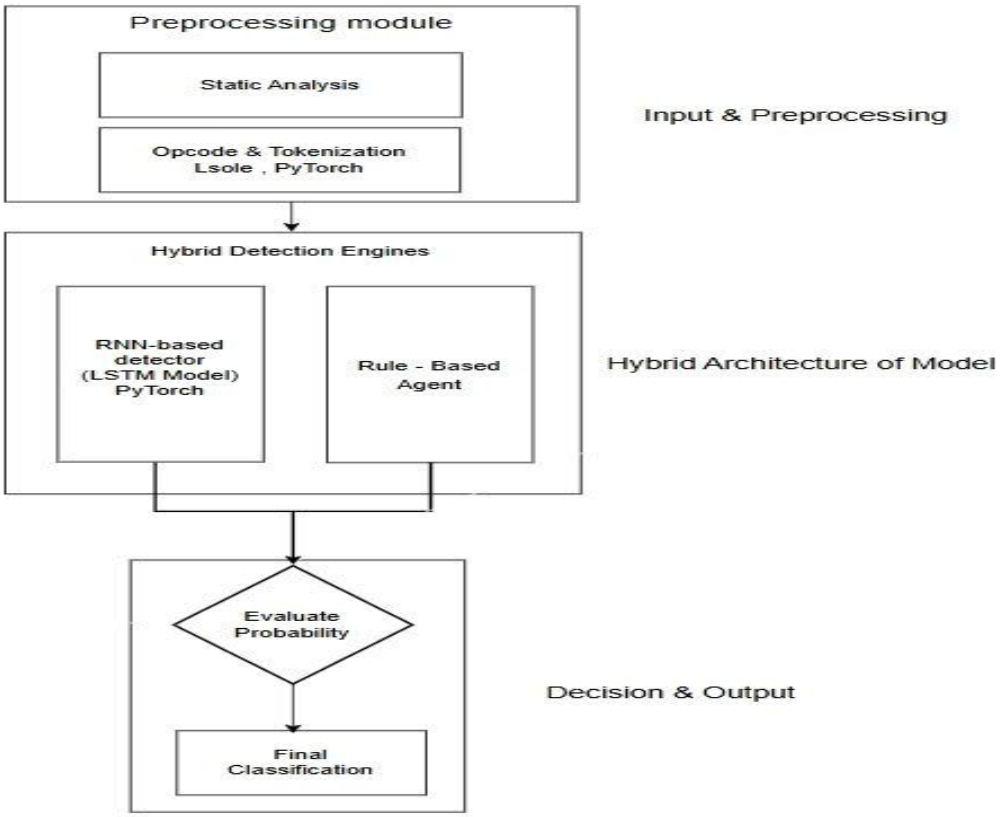

Figure 2. Recurrent neural networks (RNNs)

4.2 Graph Neural Network (GNN) Architecture

The GNN model is designed to process graph-based smart contract representations, such as ASTs or CFGs. The focus of the modelling lies in learning from structural and relational information rather than sequential order.

Architecture consists of the following stages:

- Input Graph Construction: Each smart contract is converted into a graph where nodes represent code components, such as functions, statements, or variables, and edges denote data flow or control dependencies.
- Node Embedding Layer: Each node is initialized with a feature vector that describes its attributes in the code.
- Graph Convolution or Message Passing: In this step, every node communicates with the neighbouring nodes for learning contextual and relational dependencies.
- Global Pooling Layer: It aggregates information from all nodes into a single coherent representation of the entire contract.
- Dense Layer and Output: The global feature vector undergoes a dense layer, which outputs a classification score using a sigmoid activation function.

The GNN architecture empowers the system to capture non-linear and hierarchical relations within the code, which provides deeper semantic insights into complex contract behaviours that might be hidden from traditional models.

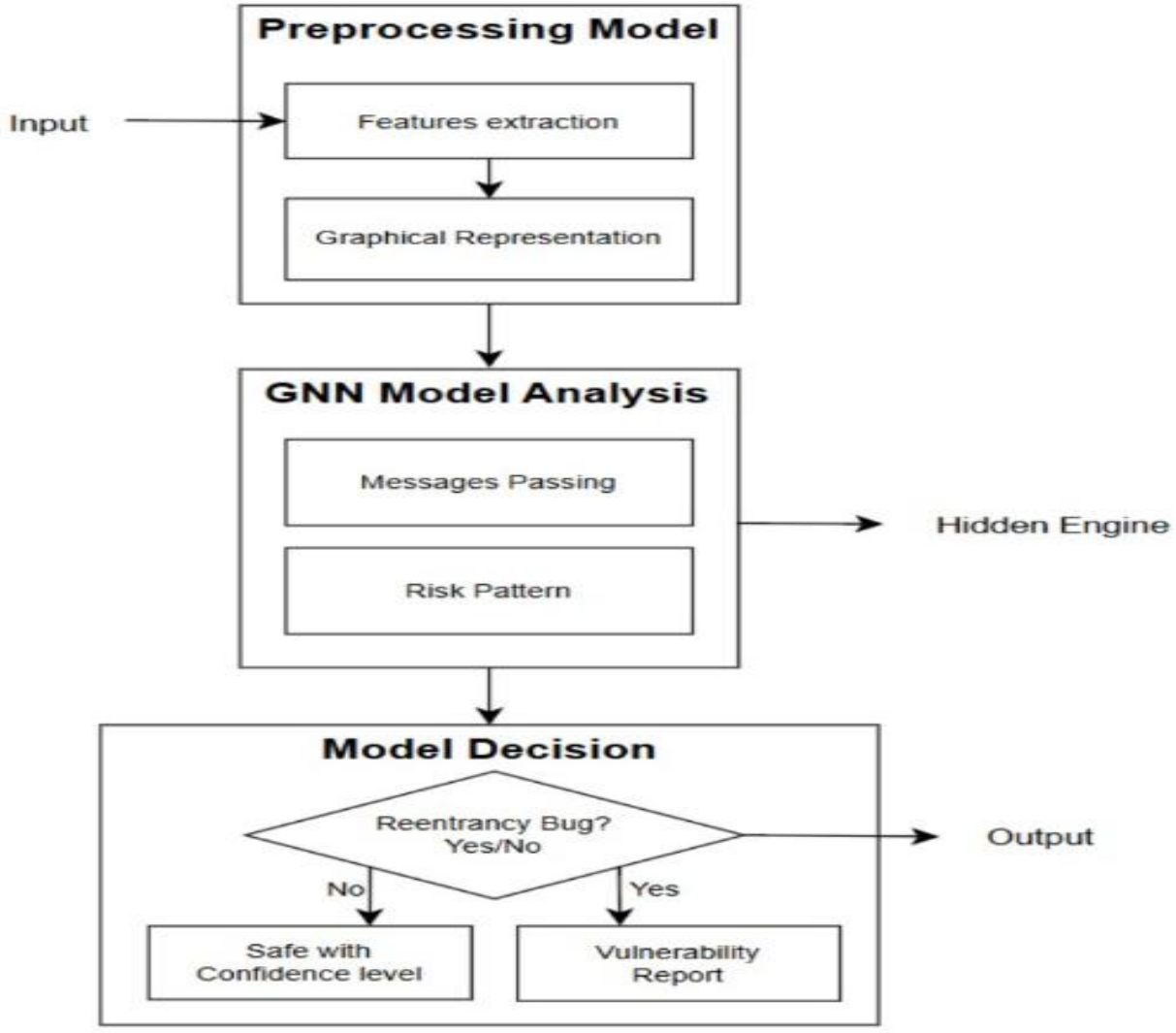

Figure 3. Graphical neural networks (GNNs)

4.3 Model Development and Optimization

Both RNN and GNN models are trained using the Adam optimizer, which adaptively adjusts learning rates for the purpose of stable convergence. The binary cross-entropy loss function is used to calculate the difference between the predicted and the actual labels. To improve model performance and reliability:

- Added dropout layers to avoid overfitting.
- Early stopping ensures that the training stops when the validation performance stops improving.
- Learning Rate Scheduling: Fine-tunes the training process for smoother convergence.

These developed architectures are implemented using the deep learning frameworks TensorFlow or PyTorch. The final trained models provide accurate predictions of vulnerabilities and lay the foundation for automated smart contract auditing.

5. Deep Q-Network based Traffic Signal control

While smart contract vulnerability detection is the main focus of this project, drawing from the concept of reinforcement learning to adaptive model optimization, as the thesis structure template requires, is done by analogy in this subsection. The Deep Q-Network concept is included here to show how similar learning mechanisms can be applied in decision-making systems.

A DQN is a reinforcement learning algorithm that couples Q-learning with deep neural networks. It allows an agent to learn optimal actions through trial and error, using rewards or penalties based on its decisions. The same principle of feedback-driven learning underlies the optimization strategy utilized in the proposed deep learning models.

6. Evaluation of Models

The performance of the deep learning models proposed, namely RNN and GNN, is evaluated with the aim of measuring their accuracy, reliability, and capability in the detection of reentrancy vulnerabilities in Ethereum smart contracts. The various models trained will be tested on unseen data from the PrimeSmartVuln dataset and their performance will be analysed based on standard classification metrics.

6.1 Evaluation Metrics

- Accuracy: It measures the full overall correctness of the model by just calculating the proportion of the total predictions that were actually correct. It provides a basic general idea of the performance of the model, but may be reliable if one class dominates the dataset or it is the overall correctness of the classification, measured as a ratio of correctly classified contracts to the total number.

Formula: $$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad 6.1$$

Where:

- TP: True Positives, TP in which all the vulnerable contracts are being classified correctly as vulnerable.
- TN: True Negatives, TN in which the safe contracts are correctly classified as safe.
- FP: False Positives, FP in which the safe contracts are incorrectly classified as vulnerable.
- FN: False Negatives, FN in which the vulnerable contracts are incorrectly classified as safe.
- Precision: It tells that how many of the contracts that were been predicted as vulnerable are actually vulnerable. It us used to measure the accuracy of all positive (vulnerable) predictions or it shows how many of the predicted contracts vulnerable are indeed vulnerable, hence reflecting the model's performance on minimizing false positives.

Formula: $$\text{Precision} = \frac{TP}{TP + FP} \quad \text{--- 6.2}$$

Interpretation: A good high precision (0.7622) meaning that the RNN model produces a very few false positives, but by correctly identifying the genuine and correct vulnerabilities most of the time

- Recall: It is used to measure that how effectively the model detects all the actual vulnerabilities. It mainly indicates the proportion of all the true vulnerabilities that were correctly identified by the model. measures the ratio at which the model detects vulnerable contracts among actual vulnerable samples and reflects the sensitivity of the model.

Formula: $$\text{Recall} = \frac{TP}{TP + FN} \quad \text{--- 6.3}$$

Interpretation: A recall of approximately 0.6912 meaning that about 69% of vulnerable contracts were actually identified correctly, although some were missed which indicates the false negatives, which can generate a real-world risk.

- F1-Score

The F1-score is being calculated as the harmonic mean of the precision and recall. It mainly provides a balanced evaluation metric when there is an uneven class distribution of trade-off between the false positives and the false negatives.

Formula: $$\text{F1-Score} = 2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}$$ --- 6.4

Interpretation: The F1-score of about 0.7249 (72 .49%) shows that the model is being effectively balances between the precision by minimizing all the false positives and the recall by correctly detecting the actual vulnerabilities.

After training and optimization of the models, these were later tested on the reserved test dataset. The following are the results:

Table 3.5.7.1 Performance Analysis

Performance Metric	RNN Model	GNN Model
Accuracy	84.11%	81%
Precision	76.22%	62% (Reentrancy), 92% (Clean)
Recall	69.12%	82% (Reentrancy), 81% (Clean)

In contrast, the RNN model had a better balance of precision and recall. It was particularly effective at determining complex structural dependencies within contracts and, therefore, only a few safe contracts were misclassified.

V. FINDINGS

The overall evaluation suggests that both models contribute significantly to enhancing the accuracy and efficiency of smart contract vulnerability detection. The GNN model, with its structural learning capability, demonstrated a more balanced and reliable performance, whereas the RNN model captured complex sequential dependencies effectively. The integration of these models within the hybrid framework promises a reliable, scalable, and automated approach towards ensuring the security of Ethereum-based dApps.

1. Report Generation Through Web Application

The final stage of the proposed system, after successful model training and evaluation, is to deploy the trained deep learning models into a web-based application. The platform can be used by developers and auditors for real-time vulnerability analysis in smart contracts, supported with comprehensive security reports.

The web application serves as an interface between the user and the trained models. It is designed to make the task of smart contract auditing more user-friendly by enabling users to directly upload Solidity (.sol) files for analysis.

Once a file is uploaded, the system processes it through the following stages:

i. Upload Code:

The user submits the smart contract via the web interface.

ii. Backend Processing:

This contract is then sent to the backend using a RESTful API, which pre-processes the text by tokenization, normalization, and graph generation.

iii. Model Execution:

Both RNN and GNN models take the preprocessed data as input. Each model independently analyses the contract and generates a probability score of the presence of a vulnerability.

iv. Hybrid Decision Mechanism:

The results from the models are combined with rule based analysis that checks for patterns of known vulnerabilities, such as improper function calls or missing state updates.

v. Report Generation:

The system thereafter compiles these results into a detailed security report highlighting the probability of the vulnerability, identified risks, and recommended preventive measures.

2. Report Structure

Key information included in the generated report is:

- Contract Information: Name, size, and version of the Solidity compiler.

- Model Results: Probability score from RNN and GNN models indicating the likelihood of vulnerability.
- Rule based Analysis: Identifies security patterns or red flags using static rule-based analysis.
- Final Classification: A merged decision Vulnerable or Safe from deep learning and rule based methods.
- Recommendation Section: The suggested corrections or design practices to eradicate the vulnerabilities detected, such as applying the Checks-Effects-Interactions pattern to avoid reentrancy.

3. System Features

The web application supplies several important features:

- Real-Time Detection: Immediate scanning and analysis of uploaded smart contracts.
- Hybrid Verification: Combining both static, rule based approaches and deep learning for reliable results.
- Visual Feedback: Graphical representation of vulnerability scores and detailed comparison between models.

4. Implementation Details

The web application is built using modern web technologies such as:

- Frontend: HTML, CSS, and JavaScript frameworks were used in the development for having a responsive interface.
- Backend: Implemented in Python, using either Flask or Django, integrating the deep learning models via REST APIs.
- Database: Used for storing the contract details, scan results, and historical reports for future reference.
- Model Integration: Using frameworks like TensorFlow or PyTorch, serialise the RNN and GNN models that have been trained, loading them dynamically during analysis.

This architecture provides for efficient communication between a user interface and the back-end model in a secure and seamless manner.

5. Performance of RNN Model

A functional hybrid framework for identifying reentrancy vulnerabilities in the smart contracts has been established by the advancements made thus far. The integration of both the deep learning and the static rule based approaches ensures that the system captures the vulnerabilities from a complementary perspective. The RNN model excels at detecting the complex sequential patterns in the smart contracts that are difficult to capture with the static rules, while the rule based checker ensures that explicit, and the well-known attack patterns are reliably identified and are not missed.

The double-layered detection of the mechanism has already proven so effective in an initial testing, in which the system generating the structured reports that include both, the probabilistic assessments and the deterministic flags. Such hybridization not only improves detection accuracy but it also enhances interpretability, as the rule based outputs provide a human- readable justification for potential vulnerabilities. Overall, the results confirm the feasibility of combining the machine learning with the traditional static analysis to strengthen the smart contract auditing. The performance of the RNN model is been evaluated using the standard classification metrics including some of the performance parameters that is accuracy, precision, recall, and F1-score.

- Accuracy (84.11%): The overall accuracy of the model was observed to be 84.11%, indicating that the majority of smart contracts were correctly classified as either non- vulnerable (safe) or vulnerable (not safe). While accuracy provides a general measure of correctness, it can be misleading in the cases of class imbalance, which makes the additional metrics highly and mostly important.
- Precision (0.7622): It is a performance parameter that shows how many of the contracts are to be predicted as vulnerable that are actually vulnerable. A precision of about 76% indicates that when the model states a contract as vulnerable, it is correct in most cases, though there remains a margin of the false positives.
- Recall (0.6912): Recall shows that how well the model will be detecting the vulnerable smart

contracts. A recall of about 69% results that the model successfully finds the majority of the vulnerable smart contracts but still misses a few, which may lead to the false negatives. The false negatives are the cases where the vulnerable smart contracts are mistakenly classified as safe are mainly risky because they can leave the system open to attack when it comes to the security of the smart contracts.

- F1-Score (0.7249): The F1-score, is being calculated as the harmonic mean of the precision and the recall, is 0.7249 which is about 72%, which means that the model maintains a fairly good balance between the correctly identifying vulnerabilities i.e. the recall and minimising false alarms i.e. the precision that reflects the well-rounded performance overall. Because of which it suggests that the model maintains a reasonable reliability while avoiding an over-optimization toward a single metric.

6. Performance of GNN Model

Building on the effectiveness of hybrid detection framework, the Graph Neural Network (GNN) model has implemented to analyse the structural and the relational dependencies within the Ethereum smart contracts. Unlike the RNN model, which focuses on the sequential opcode patterns, the GNN captures an inter-function relationships, control flow dependencies, and the inter-contract interactions that are crucial for identifying the complex reentrancy vulnerabilities often missed by the sequential models. In the integration of the GNN model into the hybrid system enhances the framework's capability to reason about the graph-structured data. This allows the model to detect the vulnerabilities based on the contextual and the relational patterns rather than just linear instruction sequences. The output of the model gives both the probabilistic predictions and the rule based confirmation signals, ensuring that the results are both data driven and interpretable. During the time of evaluation, the GNN model has been tested on the benchmark dataset of the labelled smart contracts. The model's performance is being measured by using the standard classification metrics which considers the accuracy, precision, recall, and the F1 score.

- Accuracy (81%): The overall accuracy that has been observed by the GNN model is approx 81%, which mainly highlights that the majority of the smart contracts were all correctly classified as either safe i.e. non-vulnerable or unsafe i.e. vulnerable. Though the accuracy of this GNN model is slightly lower than compared to the RNN model, the detailed metrics highlight important trade-offs in detection capability.
- Precision (0.62 for Reentrancy, 0.92 for Clean): The precision noticed for the GNN model indicates the proportion of contracts predicted in a class that actually belong to that class. For detection of the reentrancy, the precision of about 62% suggests that when the model flags a contract as vulnerable or unsafe, it is correct in most of the cases but still it has a considerable margin of the false positives. Conversely, clean contracts are highly precise about 92%, which means that the safe contracts are rarely been misclassified as vulnerable.
- Recall (0.82 for Reentrancy, 0.81 for Clean): The recall that is being measured by the GNN model measures how properly the model captures all the instances of a class. The recall of approx 82% for the reentrancy attack shows that the GNN model successfully identifies the majority of the vulnerable contracts, which is considered as highly desirable in most of the security-sensitive applications. However, about 18% of vulnerable contracts may still be missed during detection. Clean contracts also achieve balanced recall of about 81%.
- F1-Score (0.70 for Reentrancy, 0.86 for Clean): The F1-score, which is calculated as the harmonic mean of the precision and the recall, tells the complete overall balance. For the detection of the reentrancy attack, an F1-score of about 70% indicates a reasonable reliability but it also highlights that the improvements are still being possible in reducing all the false positives while maintaining the high detection rate. For the clean contracts, the F1-score of about 86% indicates a strong and a consistent performance.

7. Comparision of Models

While the RNN model gives a slightly higher overall accuracy, the accuracy measure alone did not fully indicate the model's dependability or the real-world performance. A more thorough evaluation with the

confusion matrix matrices that includes precision, recall, and F1-score indicated that the RNN model produced more false positives and false negatives. This shows that the model tended to categorize the vulnerabilities wrongly, either by issuing the false positives i.e. non- existent vulnerabilities or missing the true ones i.e. false negatives.

Comparing with the GNN model it performed a higher precision and F1 score that indicates a more balanced and reliable detection ability. Therefore, although accuracy may offer a general measure of performance, it should never be the only measure used to evaluate the models, particularly in security or detection -based problems, where misclassifications may have severe consequences

Table 4.3.1 Comparison of RNN and GNN Models

Parameter	RNN Model	GNN Model
Focus / Learning Type	Sequential pattern learning from opcode sequences	Structural and relational learning from control-flow and dependency graphs
Data Representation	Sequence-based (linear opcode instructions)	Graph-based (control flow graphs, call graphs, and inter-function
Detection Approach	Learns vulnerability signatures from sequential patterns	Learns relational vulnerability features from graph-structured data
Interpretability	Moderate — based on learned sequence features	High — rule-based signals combined with graph reasoning provide explainable outputs
Accuracy	84%	81%
Precision (Reentrancy / Clean)	0.69 / 0.94	0.62 / 0.92
Recall (Reentrancy / Clean)	0.79 / 0.85	0.82 / 0.81
F1-Score (Reentrancy / Clean)	0.73 / 0.89	0.70 / 0.86
False Positives	Slightly fewer (better precision)	Slightly higher (due to broader structural learning)
False Negatives	Slightly higher (misses some complex cases)	Lower — better at detecting complex reentrancy patterns
Computation Complexity	Lower — faster to train on sequence data	Higher — graph construction and message passing increase training time
Overall Detection Nature	Sequential vulnerability detection	Context-aware relational vulnerability detection
Best Use Case	Fast and efficient classification of opcode-level vulnerabilities	In-depth detection of complex, multi-function vulnerabilities in contracts
Performance Trade-off	High precision but slightly lower recall	High recall but slightly lower precision
Integration Role in Hybrid System	Provides temporal vulnerability patterns	Provides structural and contextual validation for final decision

VI. RESULTS AND DISSCUSSION

1. Web Application for Contract Prediction

The web application that has been developed, serves as an interactive platform that allows the users to upload the Ethereum smart contracts and predict that whether they are safe or vulnerable. It helps in providing a user-friendly interface for the model selection, result visualization, and detailed vulnerability analysis enhancing the usability for all the developers and the auditors.

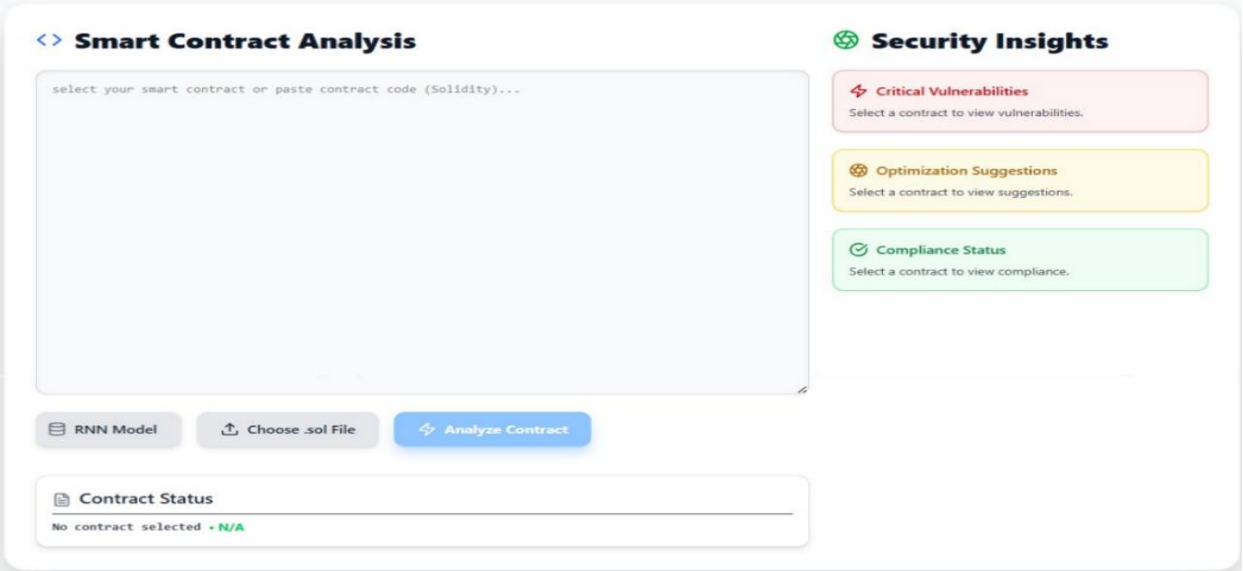

Figure 4.4.1: Web Application for Contract Prediction

2. Selecting the Smart Contract File (.sol)

The user begins the process by choosing a Solidity (.sol) file from the local system through the web interface. This file is then processed and then converted into a suitable representation for the analysis by the selected model. The system ensures that the uploaded contract is syntactically valid before further proceeding.

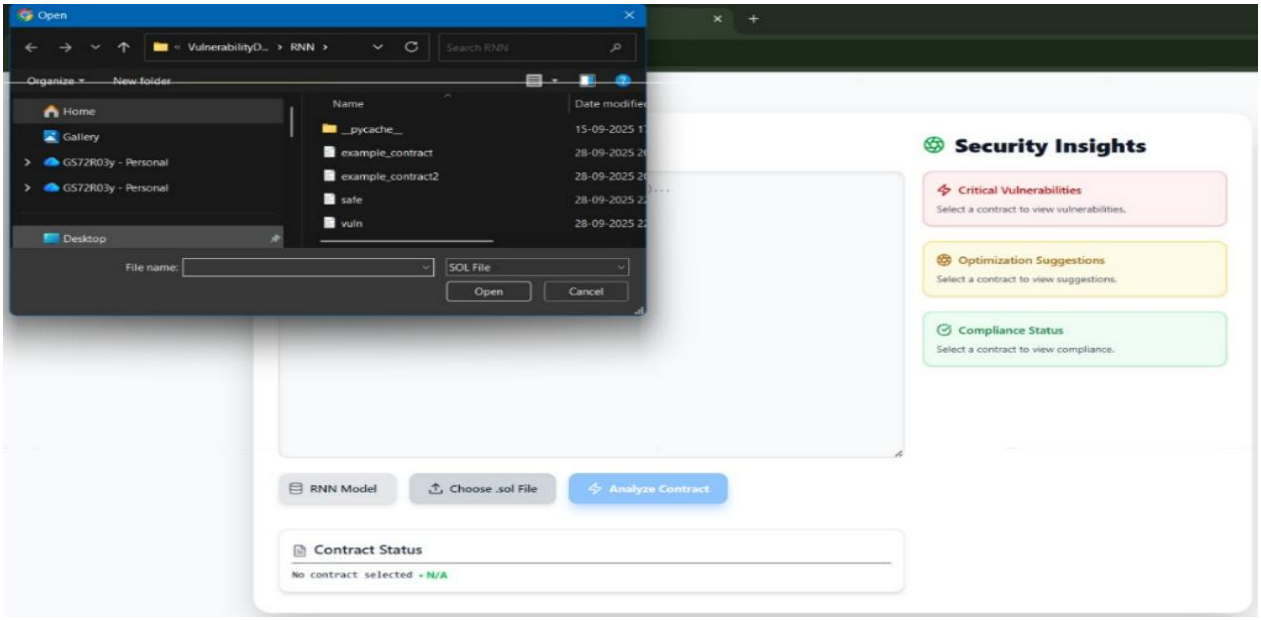

Figure 4.4.2: Selecting the Smart Contract File (.sol)

3. Analysing using the GNN Model (Vulnerable Contract)

Upon selecting the GNN model, the system firstly analyses the structural and the relational dependencies within the contract. The model helps in identifying the critical vulnerabilities, and in this case, it detects the reentrancy vulnerability. The application of this model displays a problem description, associated risk, and the required suggested fixes. The contract status flag is then set to a binary value 1, indicating that the vulnerability has been found.

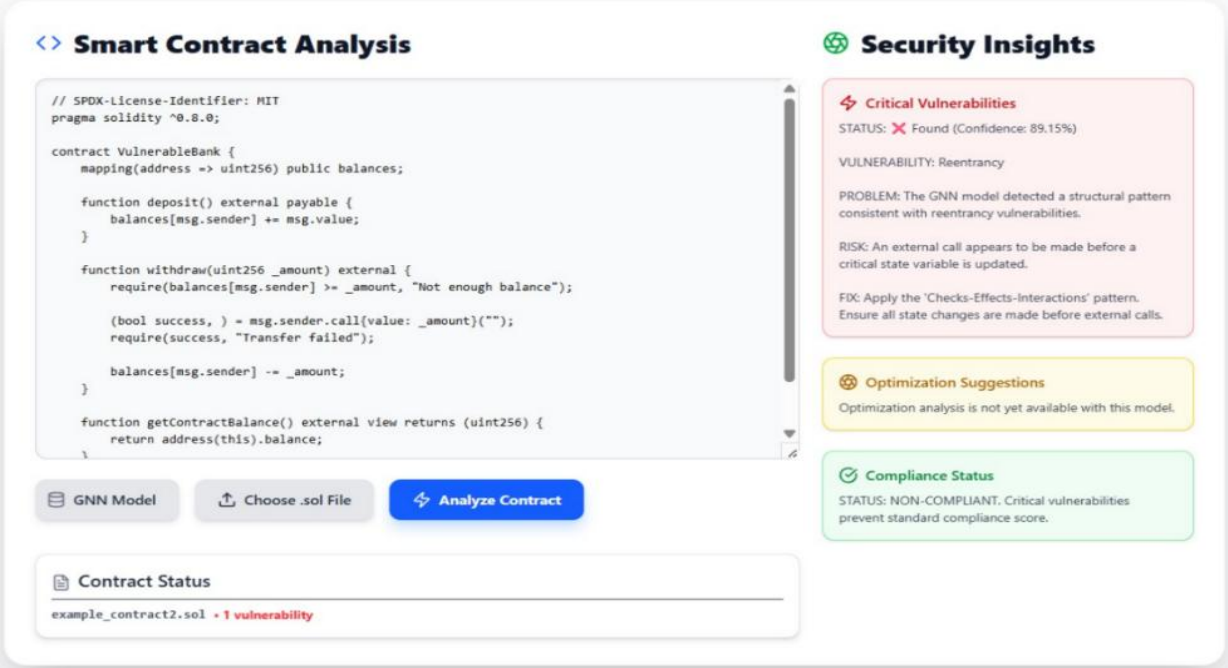

Figure 4.4.3: Analysing using the GNN Model (Vulnerable Contract)

4. Analysing using the RNN Model (Vulnerable Contract)

When the same contract, that is being used above for the GNN model, is being analysed using the RNN model, the system processes its opcode sequence patterns and also it identifies the reentrancy vulnerability. Similar to the output of the GNN model, the interface shows the risk details, probable cause, and the recommended fix that should be made, along with the binary flag = 1, confirming that the contract is being vulnerable.

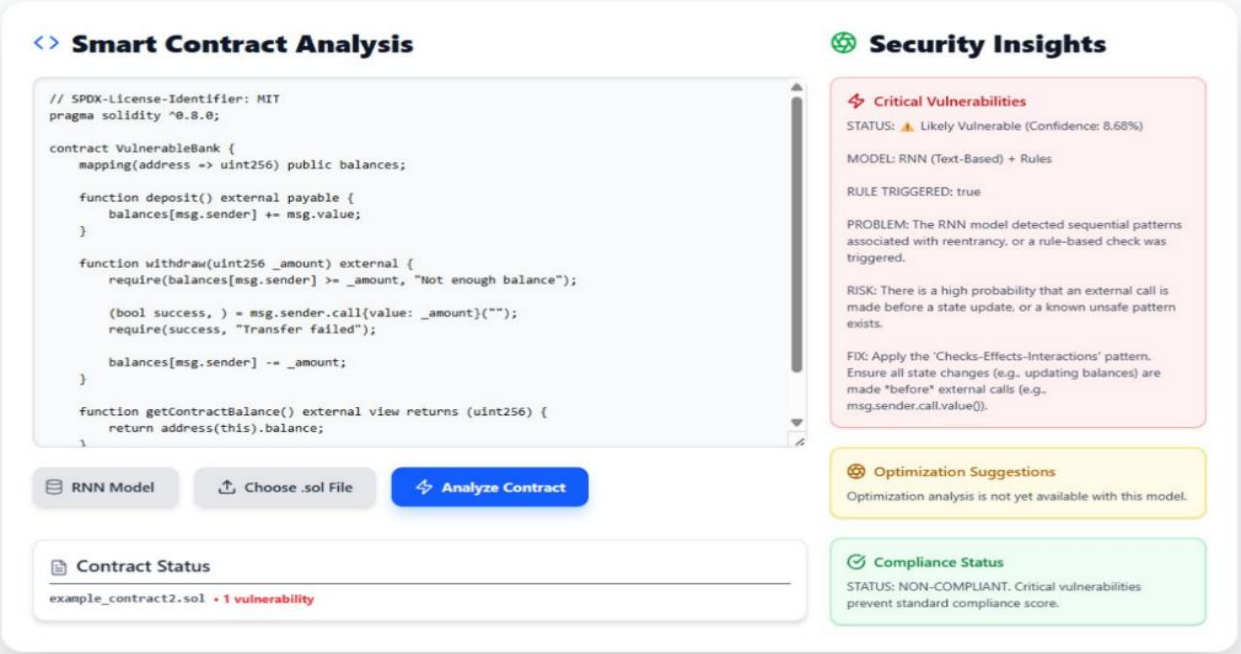

Figure 4.4.4: Analysing using the RNN Model (Vulnerable Contract)

5. Analysing using the GNN Model (Safe Contract)
Now one another contract file is being selected and is analysed through the GNN model. The model performs a graph-based reasoning and also determines that the contract is being safe with no reentrancy vulnerabilities detecting. Consequently, the critical vulnerabilities panel shows that no issues has been detected, and the contract status flags is set to a binary value of 0, indicating that no vulnerability is being found.

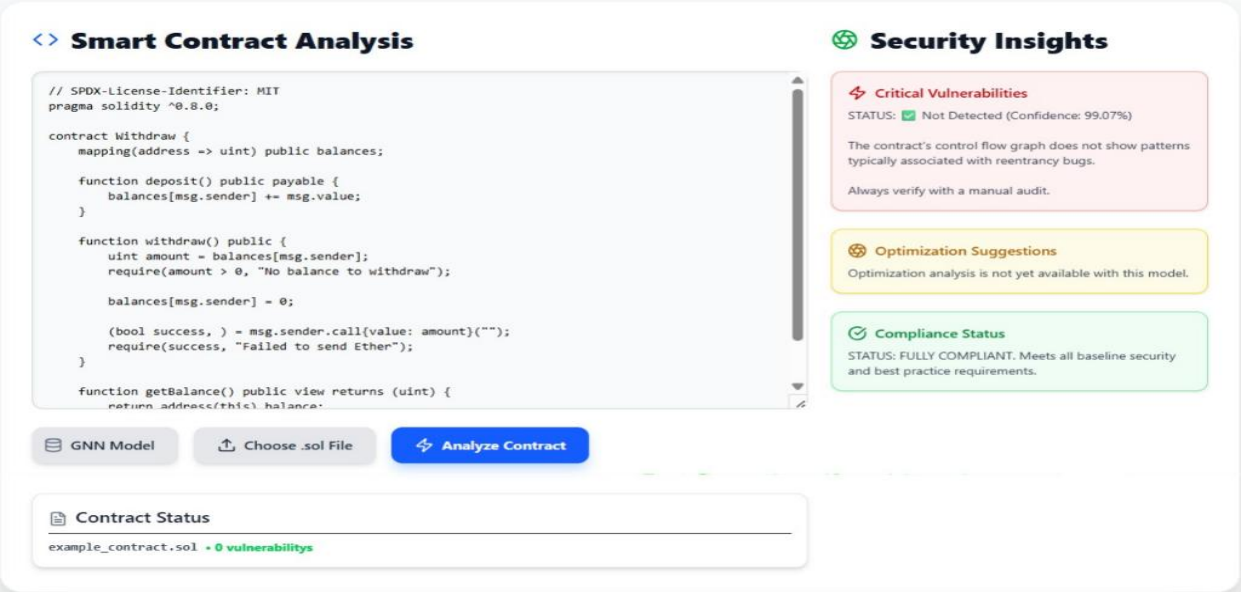

Figure 4.4.5: Analysing using the GNN Model (Safe Contract)

6. Analysing using the RNN Model (Safe Contract)
Finally, the same contract (safe one which is used above) is being analysed using the RNN model. The sequential analysis confirms that there is no such intrinsic vulnerabilities exist within the code sequence. The interface again reflects no risk or fixes required, with the binary status flag = 0, verifying that the contract is being secure.

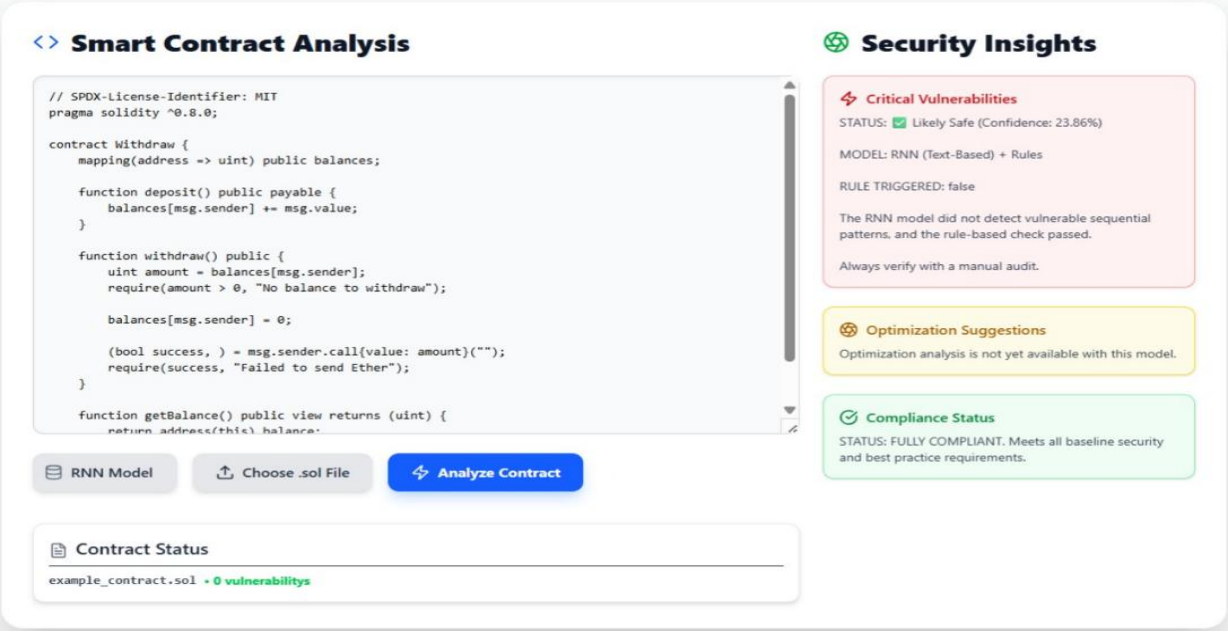

Figure 4.4.6: Analysing using the GNN Model (Safe Contract)

7. Comparative analysis of Traditional and Proposed Methods

The traditional methods that are being for detecting the smart contract vulnerabilities, such as the static analysis, symbolic execution, and the fuzz testing, primarily rely on the predefined rules and patterns matching to identify the known attack signatures. While these traditional techniques are being effective in detecting the common vulnerabilities, they often fail to recognize the newly emerging and trending or complex attack behaviours like the reentrency, which then results in the high false positives and missed detections. In the contrast, the proposed deep learning-based hybrid approach which integrates the models like the Recurrent Neural Networks (RNN) and another one is the Graph Neural Networks (GNN) to learn from both the sequential opcode patterns and the structural relationship within the smart contract. This combination mainly enables the system to adapt to the evolving vulnerabilities, provide a real-time analysis, and also improve the overall detection accuracy. Furthermore, the hybrid framework also enhances the interpretability by coupling the machine learning predictions with the rule based reasoning, while ensuring that the results are both accurately and explainable, by addressing the major limitations of the traditional detection techniques.

Table 4.4.2 Comparative Analysis of Traditional and Proposed Methodologies

Parameter	Analysis Type	Automation Level	Capability to Detect Unknown Vulnerabilities	Scalability & Efficiency	Accuracy & False positives	Adaptability to Evolving Attacks
Static Analysis Tools	Static (code – level, rule based)	Low -depends on manual rules	Law — limited to predefined rules	High for small contracts, limited for complex ones	Moderate — prone to false positives	Low
Symbolic Execution & Formal Verification	Static (path-based, mathematically verified)	Moderate - Requires expert constraints	Low -to Moderate — predefined Iogic only	Low — Computationally expensive	High accuracy but low scalability	Low
Fuzz Testing (Dynamic Analysis)	Dynamic (execution-based)	Moderate — Automated input generation	Moderate — depends on test coverage	Moderate —high resource consumption	Moderate — input dependent	Moderate
Machine Learning based detection	Data -Driven (Learned patterns)	High – learns automatically from data	High – can generalize unseen patterns	High — scalable with data	High – depends on dataset quality	High
Hybrid Static-Dynamic Approaches	Combined (Static + Dynamic)	High – Semi Automated	High detects runtime + code-based issues	Moderate — higher Overhead due to dual analysis	High — improved precision	High
Proposed Deep Learning Framework (RNN & GNN)	Hybrid (Static + Deep Learning)	Very High — fully automated, self-learning	Very High — learns structural + contextual code patterns	High —optimized neural models with adaptive learning	Very High —combines static and dynamic learning to reduce false positives	Very High — Adaptable to navel reentrancy patterns

VII. CONCLUSION

The research gives a robust deep learning framework for detecting the reentrancy vulnerabilities in the Ethereum smart contracts, that are among the most critical and the commonly exploited weaknesses in the blockchain systems [4][11]. By combining the neural

network models of which one is the Recurrent neural networks (RNNs) and another one is the Graph neural networks (GNNs) [17][25] with the static rule based analysis, the system can capture both the complex and the sequential patterns and explicit vulnerability signatures, which effectively overcome the limitations of the traditional methods such as the static analysis, the symbolic execution, and the fuzzing. An extensive testing on the PrimeSmartVuln dataset has confirmed the effectiveness of the frameworks. The RNN model achieved an accuracy of 84%, with a 76.22% of precision rate, recall of approximately 69.12%, and an F1-score of over 72.49%, which shows a good balance between correctly identifying the vulnerable contracts and keeping the false positives and the false negatives at a lower rate. Additional case studies- which includes the secure contracts that
follow the Checks-Effects-Interactions patterns and the vulnerable contracts like VulnerableBank- demonstrated that the system can reliably distinguish the safe contracts from those at risk of reentrancy attacks. A series of comprehensive experiments were carried out using the PrimeSmartVuln dataset to verify the performance of the proposed framework. The GNN model achieved 81% of accuracy, with a 62% rate of precision, a recall rate of more than 82%, and an F1-score of about 70%, which shows its strong ability to identify the vulnerable smart contracts while keeping a stable result for the secure ones. Also, the case studies for both secure contracts like those adhneural network models ering to the Checks-Effects-Interactions pattern and vulnerable contracts like the vulnerableBank further validated the effectiveness of the framework in properly separating the safe smart contracts from the ones being exposed to the reentrancy attacks. Experimental findings indicate that the increased [38] accuracy by itself does not lead to an improved or more dependable model, where an exact identification of the vulnerabilities is essential in the situations, precision, recall, and F1-score provide a better indication of the model's performance. An examination of the neural network models i.e. the Recurrent neural network (RNN) and another one is the Graph neural network (GNN) models reveal that the GNN has a more balanced precision and recall, rendering it more realistic and reliable in application to real-world vulnerability detection, with fewer false alarms and higher deployment trustworthiness.

REFERENCES

[1] Z. Yang, W. Zhu and M. Yu, "Improvement and Optimization of Vulnerability Detection Methods for Ethernet Smart Contracts," in IEEE Access, vol. 11, pp. 78207-78223, 2023, doi: 10.1109/ACCESS.2023.3298672.

[2] W. Zhang et al., "Combatting Front-Running in Smart Contracts: Attack Mining, Benchmark Construction and Vulnerability Detector Evaluation," in IEEE Transactions on Software Engineering, vol. 49, no. 6, pp. 3630-3646, 1 June 2023, doi: 10.1109/TSE.2023.3270117.

[3] Q. Zhou, K. Zheng, K. Zhang, L. Hou and X. Wang, "Vulnerability Analysis of Smart Contract for Blockchain-Based IoT Applications: A Machine Learning Approach," in IEEE Internet of Things Journal, vol. 9, no. 24, pp. 24695-24707, 15 Dec.15, 2022, doi: 10.1109/JIOT.2022.3196269.

[4] L. S. H. Colin, P. M. Mohan, J. Pan and P. L. K. Keong, "An Integrated Smart Contract Vulnerability Detection Tool Using Multi-Layer Perceptron on Real-Time Solidity Smart Contracts," in IEEE Access, vol. 12, pp. 23549-23567, 2024, doi: 10.1109/ACCESS.2024.3364351.

[5] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu and X. Wang, "Combining Graph Neural Networks with Expert Knowledge for Smart Contract Vulnerability Detection," in IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 2, pp. 1296-1310, 1 Feb. 2023, doi: 10.1109/TKDE.2021.3095196.

[6] T. -T. -H. Le, J. Kim, S. Lee and H. Kim, "Robust Vulnerability Detection in Solidity-Based Ethereum Smart Contracts Using Fine-Tuned Transformer Encoder Models," in IEEE Access, vol. 12, pp. 154700-154717, 2024, doi: 10.1109/ACCESS.2024.3482389

[7] L. Li, Y. Liu, G. Sun and N. Li, "Smart Contract Vulnerability Detection Based on Automated Feature Extraction and Feature Interaction," in IEEE Transactions on Knowledge and Data Engineering, vol. 36, no. 9, pp. 4916-4929, Sept. 2024, doi: 10.1109/TKDE.2023.3333371.

[8] Y. Cao, F. Jiang, J. Xiao, S. Chen, X. Shao and C. Wu, "SCcheck: A Novel Graph-Driven and Attention- Enabled Smart Contract Vulnerability Detection Framework for Web 3.0 Ecosystem," in

IEEE Transactions on Network Science and Engineering, vol. 11, no. 5, pp. 4007-4019, Sept.-Oct. 2024, doi: 10.1109/TNSE.2023.3324942.

[9] Y. Zhang et al., "An Efficient Smart Contract Vulnerability Detector Based on Semantic Contract Graphs Using Approximate Graph Matching," in IEEE Internet of Things Journal, vol. 10, no. 24, pp. 21431-21442, 15 Dec.15, 2023, doi: 10.1109/JIOT.2023.3294496.

[10] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur and H. -N. Lee, "Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract," in IEEE Access, vol. 10, pp. 6605-6621, 2022, doi: 10.1109/ACCESS.2021.3140091.

[11] Z. A. Khan and A. S. Namin, "A Survey of Vulnerability Detection Techniques by Smart Contract Tools," in IEEE Access, vol. 12, pp. 70870-70910, 2024, doi: 10.1109/ACCESS.2024.3401623.

[12] He, R. Wu, X. Li, S. Chan and M. Guizani, "Detection of Vulnerabilities of Blockchain Smart Contracts," in IEEE Internet of Things Journal, vol. 10, no. 14, pp. 12178-12185, 15 July15, 2023, doi: 10.1109/JIOT.2023.3241544.

[13] P. Qian et al., "Demystifying Random Number in Ethereum Smart Contract: Taxonomy, Vulnerability Identification, and Attack Detection," in IEEE Transactions on Software Engineering, vol. 49, no. 7, pp. 3793-3810, July 2023, doi: 10.1109/TSE.2023.3271417.

[14] Z. Liu, M. Jiang, S. Zhang, J. Zhang and Y. Liu, "A Smart Contract Vulnerability Detection Mechanism Based on Deep Learning and Expert Rules," in IEEE Access, vol. 11, pp. 77990-77999, 2023, doi: 10.1109/ACCESS.2023.3298048.

[15] S. -J. Hwang, S. -H. Choi, J. Shin and Y. -H. Choi, "CodeNet: Code-Targeted Convolutional Neural Network Architecture for Smart Contract Vulnerability Detection," in IEEE Access, vol. 10, pp. 32595-32607, 2022, doi: 10.1109/ACCESS.2022.3162065.

[16] Pande, S.P., Khandelwal, S., Yenurkar, G.K., Wajgi, R.D., Nyangaresi, V.O., Hajare, P.R., & Agarkar, P.T. (2025). HybridLSTM: "An Innovative Method for Road Scene Categorization Employing Hybrid Features". Computers, Materials & Continua, 84(3), 5937–5975. https://doi.org/10.32604/cmc.2025.067214

[17] Khekare, G., Yenurkar, G., Turukmane, A.V., Ameta, G.K., Sharma, P., & Phulre, A.K. (2025). "Artificial Intelligence Algorithms for Better Decision-Making". In Multi-Criteria Decision-Making and Optimum Design with Machine Learning (pp. 252–262). CRC Press.

[18] Nyangaresi, V.O., & Yenurkar, G.K. (2024). "Anonymity Preserving Lightweight Authentication Protocol for Resource-Limited Wireless Sensor Networks". High-Confidence Computing, 4(2), 100178. https://doi.org/10.1016/j.hcc.2024.100178

[19] Nyangaresi, V.O., AlRababah, A.A., Yenurkar, G.K., Chinthaginjala, R., & Yasir, M. (2024). "Anonymous Authentication Scheme Based on Physically Unclonable Function and Biometrics for Smart Cities". Engineering Reports, e13079. https://doi.org/10.1002/eng2.13079

[20] Assudani, P.J., Kadu, R.K., Yenurkar, G., Kumbhalkar, P.K., Ostwal, P.G., Kale, R.S., & Lala, G.G. (2024). "Testing Resiliency of AWS Services Using the Concept of Chaos Engineering". In Proceedings of the 2024 5th International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV) (pp. 352–356). IEEE. https://doi.org/10.1109/ICICV60198.2024.10450213

Yenurkar, G., & Mal, S. (2021). "Performance Analysis of Big Data Based Mining and Machine Learning Algorithms: A Review". Turkish Online Journal of Qualitative Inquiry, 12(7), 6750–6764.

[21] D. He, K. Ding, S. Chan and M. Guizani, "Unknown Threats Detection Methods of Smart Contracts," in IEEE Internet of Things Journal, vol. 11, no. 3, pp. 4430-4441, 1 Feb.1, 2024, doi: 10.1109/JIOT.2023.3299492.

[22] Y. Wang, X. Zhao, L. He, Z. Zhen and H. Chen, "ContractGNN: Ethereum Smart Contract Vulnerability Detection Based on Vulnerability Sub-Graphs and Graph Neural Networks," in IEEE Transactions on Network Science and Engineering, vol. 11, no. 6, pp. 6382-6395, Nov.-Dec. 2024, doi: 10.1109/TNSE.2024.3470788.

[23] Y. K. Sanjalawe and S. R. Al-E'mari, "Abnormal Transactions Detection in the Ethereum Network Using Semi-Supervised Generative Adversarial Networks," in IEEE Access, vol. 11, pp. 98516-98531, 2023, doi: 10.1109/ACCESS.2023.3313630.

[24] W. Lian, Z. Bao, X. Zhang, B. Jia and Y. Zhang, "A Universal and Efficient Multi-Modal Smart Contract Vulnerability Detection Framework for Big Data," in IEEE Transactions on Big Data, vol. 11, no. 1, pp. 190-207, Feb. 2025, doi: 10.1109/TBDATA.2024.3403376.

[25] H. Yang, J. Zhang, X. Gu and Z. Cui, "Smart Contract Vulnerability Detection based on Abstract Syntax Tree," 2022 8th International Symposium on System Security, Safety, and Reliability (ISSSR), Chongqing, China, 2022, pp. 169-170, doi: 10.1109/ISSSR56778.2022.00032.

[26] S. Arunprasath and A. Suresh, "A Reliable Framework for Detection of Smart Contract Vulnerabilities for Enhancing Operability in Inter-Organizational Systems," in Journal of Mobile Multimedia, vol. 20, no. 2, pp. 411-433, March 2024, doi: 10.13052/jmm1550-4646.2027.

[27] J. Chen, L. Wang and H. Zhu, "SmartGuard: Making Prediction Verifiable Through Transaction Sequences for Smart Contract Vulnerability Detection," in IEEE Transactions on Information Forensics and Security, vol. 20, pp. 6117-6132, 2025, doi: 10.1109/TIFS.2025.3580224.

[28] W. Wang, J. Song, G. Xu, Y. Li, H. Wang and C. Su, "ContractWard: Automated Vulnerability Detection Models for Ethereum Smart Contracts," in IEEE Transactions on Network Science and Engineering, vol. 8, no. 2, pp. 1133-1144, 1 April-June 2021, doi: 10.1109/TNSE.2020.2968505.

[29] C. Xu, H. Xu, L. Zhu, X. Shen and K. Sharif, "Enhanced Smart Contract Vulnerability Detection via Graph Neural Networks: Achieving High Accuracy and Efficiency," in IEEE Transactions on Software Engineering, vol. 51, no. 6, pp. 1854-1865, June 2025, doi: 10.1109/TSE.2025.3570421.

[30] J. Cai, J. Chen, T. Zhang, X. Luo, X. Sun and B. Li, "Detecting Reentrancy Vulnerabilities for Solidity Smart Contracts With Contract Standards-Based Rules," in IEEE Transactions on Information Forensics and Security, vol. 20, pp. 3662-3676, 2025, doi: 10.1109/TIFS.2025.3551535.

[31] W. Lian, Z. Bao, X. Zhang, B. Jia and Y. Zhang, "A Universal and Efficient Multi-Modal Smart Contract Vulnerability Detection Framework for Big Data," in IEEE Transactions on Big Data, vol. 11, no. 1, pp. 190-207, Feb. 2025, doi: 10.1109/TBDATA.2024.3403376.

[32] Z. Wang, J. Chen, P. Zheng, Y. Zhang, W. Zhang and Z. Zheng, "Unity is Strength: Enhancing Precision in Reentrancy Vulnerability Detection of Smart Contract Analysis Tools," in IEEE Transactions on Software Engineering, vol. 51, no. 1, pp. 1-13, Jan. 2025, doi: 10.1109/TSE.2024.3427321.

[33] L. He, X. Zhao and Y. Wang, "ReenSAT: Reentrancy Vulnerability Detection in Smart Contracts Using Semantic-Enhanced SAT Evaluation," in IEEE Transactions on Reliability, vol. 74, no. 2, pp. 2708-2722, June 2025, doi: 10.1109/TR.2024.3488814.

[34] D. Mishra and S. Phansalkar, "Blockchain Security in Focus: A Comprehensive Investigation into Threats, Smart Contract Security, Cross-Chain Bridges, and Vulnerabilities Detection Tools and Techniques," in IEEE Access, vol. 13, pp. 60643-60671, 2025, doi: 10.1109/ACCESS.2025.3556499.

[35] M. Bresil, P. Prasad, M. S. Sayeed and U. A. Bukar, "Deep Learning-Based Vulnerability Detection Solutions in Smart Contracts: A Comparative and Meta-Analysis of Existing Approaches," in IEEE Access, vol. 13, pp. 28894-28919, 2025, doi: 10.1109/ACCESS.2025.3532326.

[36] L. Duan, L. Yang, C. Liu, W. Ni and W. Wang, "A New Smart Contract Anomaly Detection Method by Fusing Opcode and Source Code Features for Blockchain Services," in IEEE Transactions on Network and Service Management, vol. 20, no. 4, pp. 4354-4368, Dec. 2023, doi: 10.1109/TNSM.2023.3278311.

[37] L. Chen, H. Wang, Y. Zhou, T. Wong, J. Wang and C. Zhang, "SmartTrans: Advanced Similarity Analysis for Detecting Vulnerabilities in Ethereum Smart Contracts," in IEEE Transactions on Dependable and Secure Computing, vol. 22, no. 4, pp. 4412-4427, July-Aug. 2025, doi: 10.1109/TDSC.2025.3547243.

[38] H. Chu, P. Zhang, H. Dong, Y. Xiao and S. Ji, "DeepFusion: Smart Contract Vulnerability Detection Via Deep Learning and Data Fusion," in IEEE Transactions on Reliability, vol. 74, no. 3, pp. 3544-3558, Sept. 2025, doi: 10.1109/TR.2024.3480010.

[39] Q. Huang, Y. He, Z. Xing, M. Yu, X. Xu and Q. Lu, "Enhancing Fine-Grained Smart Contract Vulnerability Detection Through Domain Features and Transparent Interpretation," in IEEE

Transactions on Reliability, vol. 74, no. 3, pp. 4207-4221, Sept. 2025, doi: 10.1109/TR.2025.3551356.

[40] S. Mani Rahnama, "DeFiSentinel: AI-Enhanced Decentralized Finance Architecture with Advanced Cryptographic Smart Contracts for Data Integrity and Threat Resilience," in IEEE Access, vol. 13, pp. 83107-83122, 2025, doi: 10.1109/ACCESS.2025.3568842.

[41] D. He, K. Ding, S. Chan and M. Guizani, "Unknown Threats Detection Methods of Smart Contracts," in IEEE Internet of Things Journal, vol. 11, no. 3, pp. 4430-4441, 1 Feb.1, 2024, doi: 10.1109/JIOT.2023.3299492.

[42] D. Han, Q. Li, L. Zhang and T. Xu, "A smart contract vulnerability detection model based on graph neural networks," 2022 4th International Conference on Frontiers Technology of Information and Computer (ICFTIC), Qingdao, China, 2022, pp. 834-837, doi: 10.1109/ICFTIC57696.2022.10075325.