Article
JOURNAL OF UNIVERSITY OF BABYLON
For Pure and Applied Sciences (JUBPAS)
Vol. 34 ; No.2 | 2026

Cause-to-Fix Learning: Causal Multi-Task Deep Learning for Smart Contract Vulnerability Detection and Repair

Mohammed Subhi Mugheedh [1]
1College of Computer and Software Engineering , University of Lorestan , mohamed07827009657@gmail.com, Khorramabad, Iran.

***Corresponding author email:** mohamed07827009657@gmail.com; mobile: 07827009657

التعلم من السبب إلى الحل: التعلم العميق متعدد المهام السببي لاكتشاف وإصلاح ثغرات العقود الذكية

محمد صبحي مغيض [1]
1 كلية هندسه الحاسوب والبرمجيات، جامعة لوريستان، mohamed07827009657@gmail.com، خرم آباد، إيران

Accepted: 16/4/2026 **Published:** 30/6/2026

ABSTRACT

Smart contracts in blockchain have been reported to be more susceptible to security exploits by programming. The traditional techniques for vulnerability detection have been reported to be mainly focused on vulnerability identification, but not on the causes and actions. This paper proposes a framework based on the concept of causal-aware deep learning, where the classification of vulnerabilities, prediction of the probability of exploiting the vulnerabilities, and causal attribution are combined for smart contract vulnerabilities. The techniques of learning the latent representation of the smart contract code using Term Frequency-Inverse Document Frequency (TF-IDF) embeddings and a shared neural network have been incorporated in the proposed model. In the experiment, the dataset was made up of 2217 smart contracts, which were categorized into four classes of vulnerabilities. These classes included reentrancy, dangerous delegatecall, integer overflow, and timestamp dependency. The model was observed to achieve 0.95 in terms of overall accuracy and 0.899 in terms of macro F1 score. The F1 values were 0.99 for reentrancy, 0.93 for integer overflow, 0.91 for timestamp dependency, and 0.77 for delegatecall. The relatively lower performance for delegatecall reflects the inherent complexity of this vulnerability type, which depends on external execution context, storage alignment, and inter-contract interactions. On the other hand, the reduced performance of this type of vulnerability is considered normal due to the imbalance in the dataset, which contains only 97 nodes specific to this vulnerability. Causal analysis of the experiment showed unique relationships between the vulnerabilities and the learned causal factors. The experiment introduced the minimal fix optimization module for generating counterfactual causal modifications for reducing exploit risk while preserving the predictions made by the model. The experiment showed notable exploit probability reduction, for example, from 0.2596 to 0.0422 and from 0.1121 to 0.0272 for representative samples. Also, the casual analysis results show that delegatecall vulnerabilities are associated with multiple moderately activated causal factors rather than a single dominant cause, increasing classification difficulty. This illustrates the difficulty in detecting this vulnerability and justifies another reason for its low classification performance. Unlike traditional systems, our method offers a new "reason to fix" model for analyzing weaknesses.

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

Background:
Smart contracts in blockchain systems are increasingly vulnerable to security exploits caused by programming errors. Traditional vulnerability detection techniques primarily focus on identifying vulnerabilities without analyzing their underlying causes or providing actionable fixes. This limitation motivates the development of approaches that can both detect vulnerabilities and explain the reasons behind them.
Materials and Methods:
This study proposes a causal-aware deep learning framework for smart contract vulnerability analysis that integrates vulnerability classification, exploit probability prediction, and causal attribution. The model learns latent representations of smart contract code using Term Frequency–Inverse Document Frequency (TF-IDF) embeddings combined with a shared neural network architecture. The experimental dataset consisted of 2,217 smart contracts categorized into four vulnerability classes: reentrancy, dangerous delegatecall, integer overflow, and timestamp dependency.
Results:
The proposed model achieved an overall accuracy of 0.95 and a macro F1-score of 0.899. Class-level F1 scores were 0.99 for reentrancy, 0.93 for integer overflow, 0.91 for timestamp dependency, and 0.77 for delegatecall. The causal analysis revealed distinct relationships between vulnerabilities and the learned causal factors. In addition, the proposed minimal fix optimization module generated counterfactual causal modifications that significantly reduced exploit probabilities while preserving model predictions, for example reducing probabilities from 0.2596 to 0.0422 and from 0.1121 to 0.0272 in representative samples.
Conclusion:
The proposed framework extends traditional vulnerability detection by introducing causal reasoning and minimal counterfactual fixes. This approach not only identifies vulnerabilities but also explains their causes and suggests targeted modifications, offering a novel “reason-to-fix” paradigm for improving smart contract security.
Key words:
Smart Contract Security, Vulnerability Detection, Causal Deep Learning, Multi-Task Learning, Counterfactual Analysis, Automated Vulnerability Repair, Blockchain Security .

1. INTRODUCTION

Blockchain (BC) technology has enabled the development of decentralized applications through smart contracts (SC), which are self-executing programs deployed on blockchain platforms such as Ethereum [1] [2]. Smart contracts, despite all the benefits derived from applying blockchain-based technology, have become victims of Cyber-attacks because of the possibility of suffering financial loss through vulnerabilities related to the source code of smart contracts [3]. The most famous Cyber-attacks related to smart contracts is known as the Decentralized Autonomous Organization (DAO) Hack, which took place in 2016 as a result of a reentrancy vulnerability, resulting in millions of dollars’ worth of cryptocurrency being lost [4]. Similar types of vulnerabilities, such as integer overflow [5], timestamp dependency [6], and unsafe delegate calls [7], have been known to be a threat to decentralized applications, and therefore it is important to have efficient techniques to deal with such issues.

Currently, the security of smart contracts is largely dependent on the use of static analysis tools [8], symbol-based execution [9], and traditional machine learning techniques [10] in the detection of vulnerabilities in the contracts. Although the current security techniques have been able to achieve some reasonable success in the detection of security vulnerabilities in smart

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com
مجلة جامعة بابل للعلوم الصرفة والتطبيقية مجلة جامعة بابل للعلوم الصرفة والتطبيقية

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

contracts, it is observed that the current techniques are largely based on black-box classifiers, which are able to indicate the presence or absence of a security vulnerability in the contract. This is largely the problem faced by the developers, as the detection of vulnerabilities does not provide the developers with any idea of how the problems are to be solved effectively. In addition, the current deep learning techniques are largely based on the achievement of the highest possible accuracy in the detection of vulnerabilities in the contracts, without considering the underlying causal links of the code with the vulnerabilities. To address the above problems, this paper proposes the causal-aware deep learning framework for the analysis of vulnerabilities in smart contracts, which extends beyond the analysis of vulnerabilities and enters the domain of causal attribution and fix generation. The methodology that is followed in this paper includes the following: the first aspect is the analysis of the vulnerability, which includes the classification, the probability, and the prediction, all of which are included under one single network for smart contracts. In addition, the counterfactual optimization module is introduced in order to obtain the minimum number of changes that lead to a significant decrease in the exploit probability, while the overall structure of the program is maintained. In addition, the experimental results of the proposed framework are presented using a dataset of 2217 smart contracts, while the following types of vulnerabilities are considered: reentrancy, risky delegatecall, integer overflow, and timestamp dependency. Although the number of contracts in the dataset that was used in this research is quite small with 2,217 smart contracts in total, the model has been designed with this problem in mind. In fact, because the TF-IDF embeddings are employed, the model is capable of generating an effective embedding of the input code without requiring a big training dataset. Moreover, the adopted architecture is a lightweight multi-layer perceptron with limited parameters, which is less prone to overfitting compared to deeper architectures. Finally, The multi-task learning setup further enhances generalization by jointly optimizing classification, exploit prediction, and causal attribution. Regularization techniques, including L1 sparsity and early stopping, are also employed to ensure stable training. Most importantly, the proposed framework has the potential to support the cause-to-fix paradigm, which is a significant step towards the explanation of the AI-driven security analysis of smart contracts.

2. LITERATURE REVIEWS

2.1 Dedicated Static Analyzers:

Ethainter [11] is designed for the identification of complex vulnerabilities, especially in smart contracts. This tool monitors the flow of malicious/wrong information and checks the flow of information among multiple transactions, especially when conditions for security have been skipped. This tool uses rules, which are stored as a data log, for checking the logic of smart contracts. However, this tool is limited to detecting security vulnerabilities without explaining why they occurred or suggesting solutions to fix the flaw in the smart contract's code. Ethainter successfully identified composite vulnerabilities (e.g., cross-contract reentrancy) that single-transaction analyzers miss. The tool detects vulnerabilities but does not explain causal origins or suggest fixes.

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

On the other hand, NeuCheck [12] uses the syntactical analysis method for the detection of vulnerabilities. This method is based on the use of abstract syntax trees, symbolic execution, and dependency graph analysis. The tool starts with the conversion of the source code into a syntax tree data structure. After that, the tool uses DOM4j queries for searching the syntax structure in order to generate the report for the detected security pattern. Although the tool is effective in the detection of vulnerabilities, it is highly dependent on static analysis and cannot detect the vulnerabilities that are the result of the combination of multiple contracts. This tool also is limited to detecting security vulnerabilities without explaining why they occurred or suggesting solutions to fix the flaw in the smart contract's code. There isn't a causal explanation or repair suggestion.

The SESCon [13] system focuses on the improvement of smart contract vulnerability detection based on the pattern of the vulnerabilities. The system uses the syntax tree of the smart contract for the analysis purpose. The system also uses XML Path Language (XPath) queries for the detection of the pattern of the vulnerabilities. The system is limited to the pattern of the vulnerabilities defined in the system and it is also limited to detecting security vulnerabilities without explaining why they occurred or suggesting solutions to fix the flaw in the smart contract's code. It is limited to predefined patterns; no causal analysis or fix generation.

There exists a particular framework, namely SmartScan [14], which was especially designed for the detection of vulnerabilities in smart contracts, namely Denial of Service vulnerabilities. SmartScan was designed according to the following phases: the static analysis phase, in which the smart contract is analyzed in terms of potential DoS vulnerabilities using certain patterns and transformation rules; the dynamic analysis phase; and the reporting phase. This tool is limited to detecting security vulnerabilities without explaining why they occurred or suggesting solutions to fix the flaw in the smart contract's code. There isn't explanation of vulnerability causes or automated repair.

2.2 Dynamic Analysis Tools:

The ModCon [15] is a method through which the definition of the expected behavior of smart contracts is specified using state machine models. The state machine models specify the states and transitions of the smart contracts, and the verification of the smart contract is possible using the state machine models. The method is heavily dependent on the model specification, and the specification is prone to error. Model specification is error-prone; no causal attribution.

Etherolic [16] is another security analysis tool, which is based on Ethereum Virtual Machine (EVM) bytecode, and it uses code obfuscation techniques for security analysis. The components of Etherolic include contamination engine for handling untrusted inputs, an attack indicator, a vulnerability detection engine, a log analyzer, a test generator, and a report generator. No causal explanation or fix suggestion.

sFuzz [17] is an adaptive fuzz testing tool for smart contracts, which is capable of automatically generating test inputs through the utilization of mutation strategies, as well as the monitoring of smart contracts for abnormal behaviors. It detects crashes and anomalies but does not explain why a vulnerability exists or how to fix it.

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

The SODA framework, as proposed in [18], is focused on the detection of attacks in smart contracts in real time. In this regard, the details of the execution of the transactions are gathered from the blockchain environment, and then the detection layer is used for the detection of any suspicious or malicious activity. No causal or repair information.

2.3 Hybrid and Verification Systems:

FSPVM-E, which was proposed in reference [19], utilizes different techniques such as symbolic execution, theorem proving, and static analysis to identify vulnerabilities in smart contracts. FSPVM-E has different components, namely GERM, which is a memory model, Lolisa, which is a specification language defined formally and used to create a specification using the language named Solidity, FEther, which is a formal interpreter used to execute the specification, and some tools that are auxiliary. It requires high manual effort and there isn't a causal explanation or automated fix.

SmartBugs [20] is a benchmarking environment that integrates ten tools for smart contract security analysis. SmartBugs offers one of the most comprehensive environments for experimenting with the effectiveness of various approaches to identifying vulnerabilities in smart contracts. This tool is limited to detecting security vulnerabilities without explaining why they occurred or suggesting solutions to fix the flaw in the smart contract's code. There isn't causal analysis or repair.

Smart Graph [21] is a web-based application used for creating Unified Modeling Language (UML) diagrams based on smart contract code, as discussed in. The relationship and interaction of functions and components of the smart contract are depicted in the UML diagram, and this is useful in understanding the smart contract, which is useful for debugging the smart contract. It depends on visualization only and there isn't vulnerability detection or repair.

2.4 Deep Learning/Machine Learning Methods:

The study in [22] uses Graph Neural Networks (GNN) for the representation of the SC code in the form of a graph, considering both syntactic and semantic relationships. In the representation, the major nodes of the SC code have been related to the function calls, and the secondary nodes have been related to the variables. There isn't causal explanation or repair.

In the DA-GNN [23], the graph-based approach was introduced, which uses the dual attention mechanism to improve the quality of feature extraction for the smart contract code. Although this approach is helpful in representing the key components of the smart contract code, it is computationally expensive and requires more computational power. This requires costly calculations and there is no causality.

In the SCVDIE-ENSEMBLE [24], the combination of various deep learning models, i.e., Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Deep Neural Network (DNN), is carried out under the umbrella of ensemble learning. This improves the accuracy of the results obtained from the detection system. The major disadvantage associated with this approach is the computational cost involved in the training of the models. This tool is

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

مجلة جامعة بابل للعلوم الصرفة والتطبيقية مجلة جامعة بابل للعلوم الصرفة والتطبيقية

limited to detecting security vulnerabilities without explaining why they occurred or suggesting solutions to fix the flaw in the smart contract's code. There isn't no explainability or repair.

The CodeNet [25] model is based on convolutional neural networks, and the steps involved in the model include compiling the code, converting the code into bytecode, then converting the code into a fixed-size format, and finally converting the code into an image format to be analyzed using CNN. Although this is an efficient model, there is always a possibility that information could be lost during the execution of the smart contract code, depending on the nature of the image constraint, and CNN has high complexities in terms of computation and storage. This tool is limited to detecting security vulnerabilities without explaining why they occurred or suggesting solutions to fix the flaw in the smart contract's code. Information loss due to image conversion; no causality.

The Eth2Vec [26], based on the Paragraph Vector Distributed Memory model combined with certain techniques of Natural Language Processing (NLP), has been proposed to learn the codes through the transformation of the sequences of codes into a vector representation. The model, however, could face problems in terms of the ability to generalize new vulnerabilities that were not included in the training set. Poor generalization to new vulnerability types.

The study [27] proposed a multi-task learning model with a lower sharing layer (using an attention mechanism with an Artificial Neural Network (ANN) to transform textual representation into vectors) and a Task-Specific Layer (using CNNs to classify different vulnerability types). No causal attribution or fix generation.

A model known as Multi-Layer Perceptron Artificial Neural Network (MLP-ANN), which uses features extracted from smart contracts through Control Flow Graphs, is introduced by [28]. Although this paper attempts to resolve the issue of dataset imbalance through fault injection, it is questionable whether it will be effective in handling the actual patterns of vulnerability. Relies on synthetic faults; there isn't causality.

2.5 Identified Gaps and Motivation for This Work:

From the literature reviewed above, three major gaps emerge:

1) Lack of suggestions about fixing vulnerabilities: None of the current solutions, whether they are Ethainter, sFuzz, GNN-based detector, or others, provide any follow-up actions beyond detecting the presence of vulnerability in smart contract codes. For example, none of them provide suggestions No cause-and-effect explanations provided by any existing solutions: The current solutions (either static, dynamic, hybrid or deep learning models) are regarded as being black-box or rule-based approaches. They simply identify whether there exists vulnerability in smart contract code based on certain types of exploit, such as "reentrancy detected" but without providing any information about the causes that make the code vulnerable to a particular exploit.
2) on how to fix the code.
3) Limited use of causal modeling in deep learning for smart contracts: While multi-task learning has been explored, no existing approach integrates causal attribution as a

ISSN: 2312-8135 | Print ISSN: 1992-0652 | www.journalofbabylon.com | jub@itnet.uobabylon.edu.iq | info@journalofbabylon.com

learnable component with counterfactual intervention to estimate exploit probability reduction after minimal changes.

The proposed framework addresses these gaps through the following three aspects of the design: (i) causality learning, (ii) exploiting causality in the optimization module, and (iii) mapping causality to TF-IDF features.

3. MATERIALS AND METHODS

This research proposes a multi-task causal neural model for smart contract vulnerability detection that simultaneously:

1) Classification of vulnerabilities in smart contracts
2) Predicting the probability of exploitation
3) Predicts a vector of causal factors (explainable causal probabilities).
4) The model supports optimization to calculate the optimal solution which is the smallest intervention on causal factors that reduces the probability of exploitation below a security threshold.

The proposed approach combines discriminative learning [29] (classification), causal attribution [30] (learned causal head), and constrained counterfactual optimization [31] (projected optimization over causal vector). The model is trained train, then fine-tuned to better calibrate exploitation and counterfactual behavior. Figure 1 shows the overall pipeline of the proposed methodology. The multi-task causal model utilizes the TF-IDF [32] feature and makes predictions for class, exploit probability, and causal factors. The counterfactual optimizer computes the minimum causal intervention, and links for causes are represented as features based on TF-IDF feature for suggesting code changes.

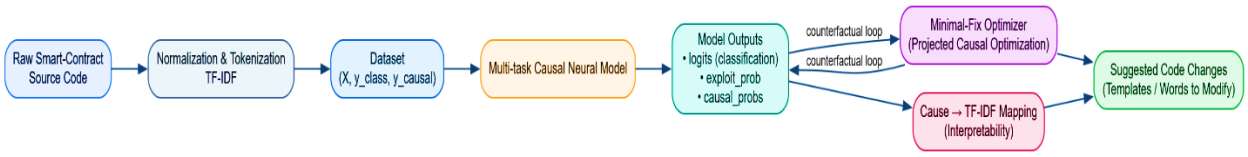

Figure 1. The overall pipeline of the proposed methodology

3.1 Data embedding and notation:

Assume that $\mathcal{D} = \{(c_i, l_i)\}_{i=1}^{N}$ is the dataset of N smart-contract where c_i is the source code for contract i and $l_i \in \{1, \dots, K\}$ is the vulnerability label. We use SC_4label dataset [33] which we obtain it from Kaggle. The dataset contains K=4 types of vulnerabilities which shown in Table 1.

ISSN: 2312-8135 | Print ISSN: 1992-0652 | www.journalofbabylon.com | jub@itnet.uobabylon.edu.iq | info@journalofbabylon.com

Table 1. The types of vulnerabilities in the dataset used

Vulnerability Type	Samples Number
Reentrancy (RE)	**1218**
Integer Overflow (OF)	**590**
Timestamp Dependency (TP)	**312**
Dangerous Delegate Call (DE)	**97**

We compute the TF-IDF embeddings as shown in formula (1):

$$x_i = TFIDF(c_i) \in \mathbb{R}^D \qquad (1)$$

Where:

$D = 256$ represents the embedding dimension (TF-IDF max_features).

3.2 Model architecture and forward pass:

The framework is based on a framework of a multi-task causal neural network to perform the following tasks simultaneously:

1) Vulnerability classification (what type of exploit exists), and
2) Causal attribution (what semantic causes are there for this exploit).

The framework is clearly separated into learning, causal, and decision processes, ensuring high accuracy in prediction and fix generation. Figure 2 shows the model architecture.

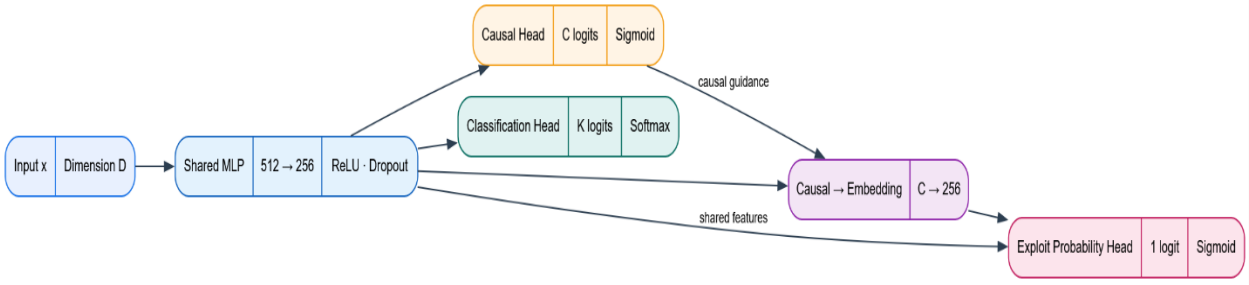

Figure 2. The model architecture

3.2.1 Shared feature backbone:

We denote the model by f_θ with parameters θ. The input embedding is first mapped into a latent representation as shown in formula (2):

$$h = f_\theta(x) \qquad (2)$$

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

Where:

$f_\theta(.)$: The shared neural backbone which is a multi-layer perceptron [34] neural network with two layers of 256 neurons.

h: A compact latent feature vector.

This shared representation is optimized jointly for classification and causal prediction.

3.2.2 Causal Head:

The causal head predicts causal contribution scores as shown in formula (3):

$$C = \sigma(W_C h + b_C) \qquad (3)$$

Where:

$C \in [0,1]^m$: The causal vector where each element C_i represents the degree to which cause i contributes to exploitability.

m: The number causal factors.

W_C: Learnable weights.

b_C: The bias.

$\sigma(.)$: The sigmoid function, ensuring interpretability as causal strengths [35].

3.2.3 Causal to embedding projection:

To allow causal inference to influence the classifier we project the causal vector back into the latent space as shown in formula (4):

$$\Delta h = \alpha . g_\emptyset(C) \qquad (4)$$

Where:

$g_\theta(.)$: Linear projection connecting causal space to latent space.

$\emptyset$: The parameters of $g_\theta(.)$.

$\alpha \in \mathbb{R}^+$: The learnable causal scaling factor.

Δh: The causal effect on the representation.

This term explains how causal factors structurally modify the decision-making process.

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

3.2.4 Causally-augmented representation:

The final representation used for classification is shown in formula (5):

$$\tilde{h} = h + \Delta h \qquad (5)$$

Where:

h: The semantic representation.

Δh: The causal effect on the representation.

3.2.5 Classification head:

The type of exploitation is shown in formula (6):

$$\hat{y} = softmax(W_y \tilde{h} + b_y) \qquad (6)$$

Where:

$\hat{y} \in \mathbb{R}^K$: The predicted class probability vector.

K: The number of vulnerability classes.

W_y: classification weights.

b_y: The bias.

The softmax ensures a valid probability distribution.

3.2.6 Exploit probability:

For fix generation, we define the exploit probability as shown in formula (7):

$$P_{exploit} = \max(\hat{y}) \qquad (7)$$

This metric measures how confident the model is that nodes are vulnerable where the high values (≈1.0) indicate severe exploitability and reducing this probability is the objective of causal intervention.

3.2.7 Counterfactual Intervention (Fix Generation):

To simulate fixes, we intervene on the causal vector as shown in formula (8):

$$C^m = C \odot m \qquad (8)$$

Where:

$m \in \{0,1\}^m$: The binary intervention mask.

$\odot$: Element-wise multiplication.

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

The masking of the causal factor corresponds to removing or mitigating that vulnerability.

The counterfactual exploit probability becomes is shown in formula (9):

$$P_{exploits}^{(m)} = f\left(x, C^{(m)}\right) \qquad (9)$$

Where x is the embedding of the contract.

This formulation answers the question: What would the exploit probability be if specific causes were fixed?

3.2.8 Optimization objective:

The optimal fix is defined as shown in formula (10):

$$m^* = argmin_m \, P_{exploits}^{(m)} \quad Where \, \|m\| \leq \varepsilon \qquad (10)$$

Where:

$\|m\|$: The Hamming distance [36] (number of changes).

ε: The threshold of fix minimality.

This formula imposes small fixes that preserve the semantics.

This architecture offers several advantages, such as:

1) Classification performance is preserved because the backbone and classifier remain unchanged.
2) Causal explanations are explicit, not post-hoc.
3) Fixes are actionable because they are based on structural interventions.
4) Alignment with causal inference theory, not correlation.
5) The model doesn't just say `This contract is vulnerable` but it says `This contract is vulnerable because of these causes, and if you fix these specific causes, exploitability will reduce`.

3.3 Loss function and training objectives:

We use a multi-task loss combining classification, exploit prediction, causal supervision (weak labels), and a small sparsity regularizer. The multi-task loss function is shown as shown in formula (11):

$$\mathcal{L} = \mathcal{L}_{cls}(z_{cls}, y) + \omega \mathcal{L}_{exp}\left(p_{exp}, y_{exp}\right) + \beta \mathcal{L}_{causal}(\hat{\alpha}, \alpha_{week}) + \gamma \frac{\sum_{j=1}^{C} |\hat{\alpha}_j| = 1}{C} \qquad (11)$$

Where:

p_{exp}: The predicted probability that a smart contract is exploitable.

ISSN: 2312-8135 | Print ISSN: 1992-0652 | www.journalofbabylon.com | jub@itnet.uobabylon.edu.iq | info@journalofbabylon.com

y_{exp}: the ground truth label for exploitability. It is defined as 1 for all vulnerable contracts.

$\mathcal{L}_{cls}$: Cross_entropy loss function [37].

$z_{cls} = W_y\tilde{h} + b_y$ which predicted $\hat{y} = softmax(W_y\tilde{h} + b_y)$.

$\mathcal{L}_{exp}$: Binary cross-entropy (BCE) [38] of exploit label y_{exp} vs exploit probability p_{exp}.

$\mathcal{L}_{causal} = \frac{1}{C}\sum_{J=1}^{C} BCE(\hat{\alpha}_j, \alpha_{weak.j})$: is a weak supervision target for causal factors (constructed from class and cause mapping) where $\hat{\alpha}_j \in [0,1]$ is the causal probability and α_{weak} is a weak label for causes based on label heuristics (e.g., reentrancy triggers certain cause bits).

ω, β, γ: hyper-parameters to detect the weight of each component of the loss function.

Joint training with causal supervision results in the formation of a causal head that is associated with meaningful properties at the code level, and supports the discovery of workarounds.

3.4 Minimal-fix optimization:

We pose the minimal-fix problem for a single sample x as an optimization over continuous causal variables $\hat{\alpha} \in [0,1]$ (the causal probabilities) with two objectives: minimize exploit probability and minimize deviation from original causal vector $\hat{\alpha}_0$ (L1 (Lasso) sparsity regularization for minimality [39]). The optimization problem is formulated as shown in formula (12):

$$\min\left(p_{exp}(x,\alpha) + \lambda\|\alpha - \hat{\alpha}_0\|\right) \qquad (12)$$

Where:

$p_{exp}(x,\alpha)$: The exploit probability when the model uses causal override α.

$\hat{\alpha}_0$: The model's original causal prediction for x.

λ: Controls sparsity (small λ allows larger changes; large λ encourages minimal edits).

$\|\alpha - \hat{\alpha}_0\|$: The Euclidean distance.

We solve this formula with projected gradient methods (Adaptive Moment Estimation (Adam)[40] + clamp to [0,1]) minimizing the surrogate as follows:

1) Start at $\alpha^0 = \hat{\alpha}_0$.
2) Use Adam optimizer to update continuous α. After each step clamp to [0,1].
3) Track the best α found (lowest p_{exp} or minimal L1 with $p_{exp} \leq \varepsilon$).
4)

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

3.5 Discrete (binary) candidate enumeration + hybrid refinement:

To increase interpretability we combine the flowing:

1) Binary Hamming enumeration: flip up to H bits of the binarized causal mask $\tilde{\alpha}_0 = 1\,[\hat{\alpha}_0 > 0.5]$ to produce candidate masks; evaluate p_{exp} on each candidate.
2) Hybrid refinement: take the top-K binary candidates and apply continuous optimization initializing at the candidate to refine.

This two-stage approach benefits interpretability (binary masks = cause removed) and optimization (continuously refines probabilities). Figure 3 shows the minimal-fix flow diagram.

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

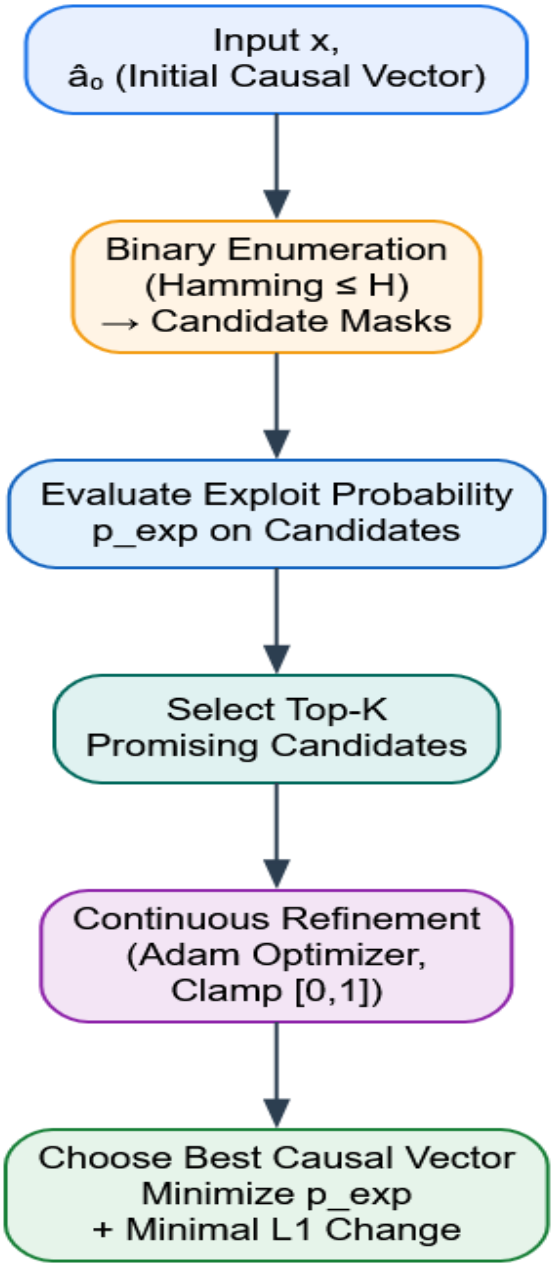

Figure 3. The minimal-fix flow diagram

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

3.6 Mapping causes to TF-IDF features (interpretability):

We map each cause to TF-IDF features by computing Pearson correlation [41] between a cause probability $\hat{\alpha}_j$ and each TF-IDF feature ϕ_k across a sampled set as shown in formula (13):

$$\rho_{j,k} = correlation(\hat{\alpha}_j, \phi_k) \qquad (13)$$

We report top positive and negative features. Positive $\rho_{j,k}$ indicates the presence of token k correlates with cause j (associated with vulnerability driver). Cause-feature correlations identify concrete lexical tokens and patterns that can be used to suggest code-level edits. Figure 4 shows the cause to feature mapping mechanism.

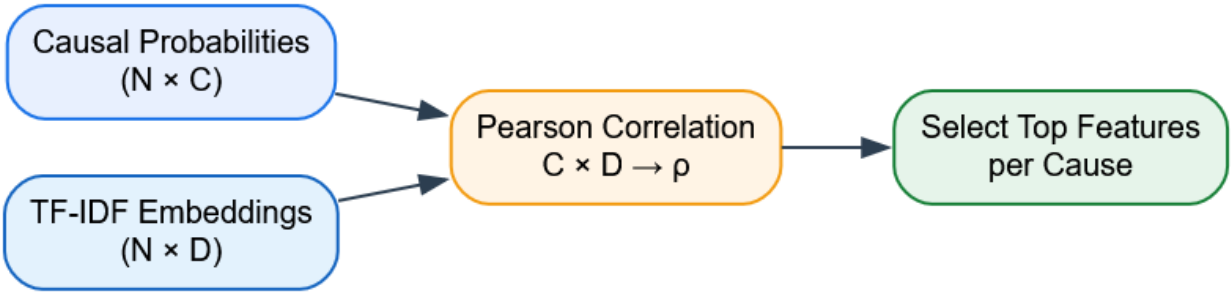

Figure 4. The cause to feature mapping mechanism

3.7 Evaluation Metrices:

To evaluate the classification performance, the confusion matrix is used. Confusion matrix is a table format used to show the performance of the model, particularly in supervised learning. The structure of the confusion matrix is illustrated in Figure 5.

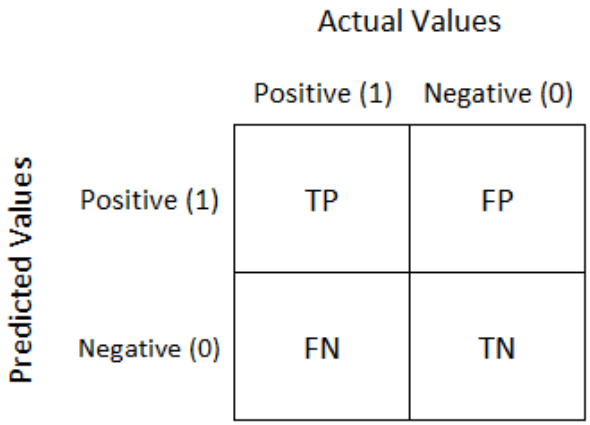

Figure. 5. The confusion matrix structure

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

Where:

- TP (True Positive): When the model correctly predicts a positive outcome.
- TN (True Negative): When the model correctly predicts a negative outcome.
- FP (False Positive): When the model incorrectly predicts a positive outcome.
- FN (False Negative): When the model incorrectly predicts a negative outcome.

These parameters are used to calculate the performance metrics (recall, precision, and accuracy) as follows:

- Recall measures the proportion of actual positives that are correctly identified by the model as shown in formula (14).

$$Re\,c\,all = \frac{TP}{TP + FN} \qquad (14)$$

- Precision measures the proportion of positive predictions that are actually correct as shown in formula (15)

$$Pr\,e\,cision = \frac{TP}{TP + FP} \qquad (15)$$

- Accuracy measures the proportion of correct predictions (both positives and negatives) out of all predictions as shown in formula (16)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (16)$$

- F1_Score is a metric that combines precision and recall, providing a balanced evaluation of a model's overall performance. F1 score is shown in formula (17).

$$F1_{Score} = 2 \times \frac{Pr\,ecision \times Re\,call}{Pr\,ecision + Re\,call} \qquad (17)$$

4. RESULTS AND DISCUSSION

This chapter is based on the discussion of the experiment results based on the proposed multi-task causal model for smart contract vulnerabilities classification and suggesting fixes. For the experiment, the proposed model was trained on a filtered dataset of 2,217 smart contracts. After preprocessing, it was able to generate 2,217 samples of four different types of smart contract vulnerabilities, namely reentrancy, dangerous delegatecall, integer overflow, and timestamp

ISSN: 2312-8135 | Print ISSN: 1992-0652 | www.journalofbabylon.com | jub@itnet.uobabylon.edu.iq | info@journalofbabylon.com

dependency. These were divided into 70%, 15%, and 15% for training, validation, and testing, respectively, i.e., 1,552, 332, and 333.

4.1 Dataset and experimental setup:

The dataset used in this research was collected from publicly available smart contract vulnerability datasets and contained 2,217 samples after applying filters on the four classes of interest. From Table 1, it is obvious that there is class imbalance in the dataset, and reentrancy is the dominant class.

The model is using TF-IDF vectors of size 256 as input feature vectors. The model is trained using the Adam optimizer with a learning rate of 3e-4, with a maximum of 60 epochs with early stopping using the macro F1 score on the validation set with patience 10. L1 sparsity regularization with a regularization strength of 0.01 is used on the causal probabilities during the training process. The fine-tuning process involves fine-tuning the exploitability head and the causal to latent space, with the classifier head frozen, using hinge losses with a large counterfactual weight of 30, with a maximum of 20 epochs with early stopping using F1 score with patience 5. The aim is to train the model such that if all causes are set to zero, the exploitability probability is low (less than 0.05), and the original prediction probability is high (greater than 0.95). The random seed was fixed to 42 to ensure reproducibility.

4.2 Classification performance:

Figure 6 shows the confusion matrix results.

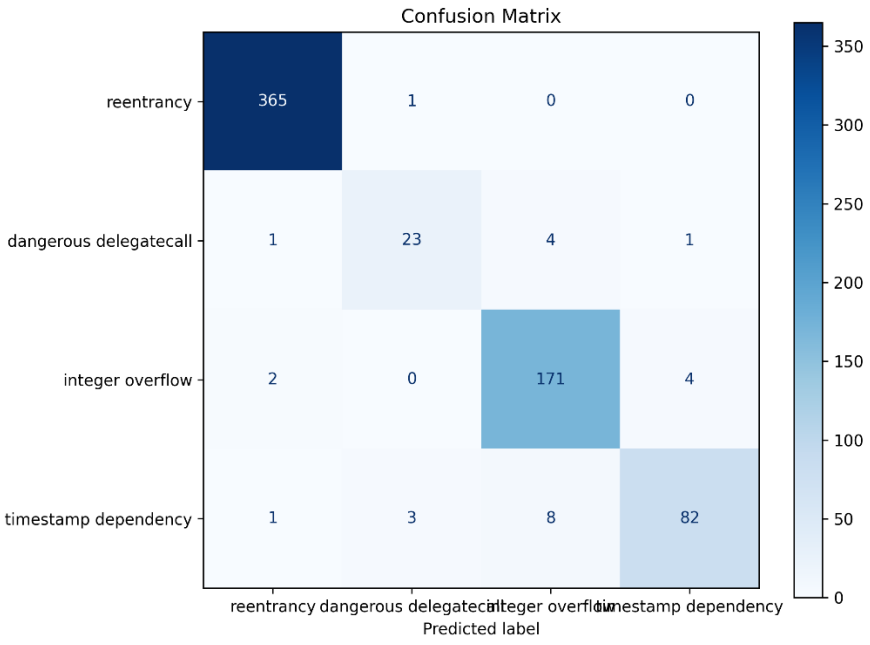

Figure 6. The confusion matrix results

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

The CM provides a snapshot of the performance. The diagonal elements represent the correct predictions for each class. For example, the top-left element (365) indicates 365 correct predictions for reentrancy class.

Off-diagonal elements represent misclassifications. For instance, the element in the second row, first column (1) indicates that 1 instances of delegatecall class were misclassified as the reentrancy class.

The model has attained a macro F1-score of 0.921 on the test set, which indicates its robustness in classifying all four types of vulnerability. Precision, recall, and F1-score of each class individually are shown in Table 2.

The model is performing extremely well on the majority class, i.e., reentrancy, with an F1-score of 0.99. It is performing extremely well on integer overflow and timestamp dependency with F1-scores of 0.96 and 0.92, respectively. It is performing reasonably on the minority class, i.e., dangerous delegatecall, with an F1-score of 0.82, considering the rarity of this class and the complexity of the delegatecall vulnerability. The F1-score is 0.97, which is a clear sign of its high accuracy.

Table 2. Classification report

Class	Precision	Recall	F1-score
reentrancy	0.99	1.00	0.99
dangerous delegatecall	0.85	0.79	0.82
integer overflow	0.94	0.97	0.96
timestamp dependency	0.94	0.89	0.92
accuracy		0.97	
macro avg	0.95	0.92	0.93
weighted avg	0.97	0.97	0.97

4.3 Effect of Fine-Tuning on Exploitability:

After fine-tuning the exploit head, the classification performance remained stable. Table 3 compares the per-class F1 scores before and after fine-tuning.

Table 3. Comparison of F1 scores before and after fine-tuning

Class	F1 before	F1 after	Δ
reentrancy	0.99	0.99	0
dangerous delegatecall	0.82	0.82	0
integer overflow	0.96	0.95	-0.01
timestamp dependency	0.92	0.91	-0.01
macro avg	0.93	0.92	-0.01
weighted avg	0.97	0.96	-0.01

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

The exploitability head, however, undergoes a dramatic change. Figure 7 shows boxplots of the predicted exploit probability for each class before and after fine-tuning.

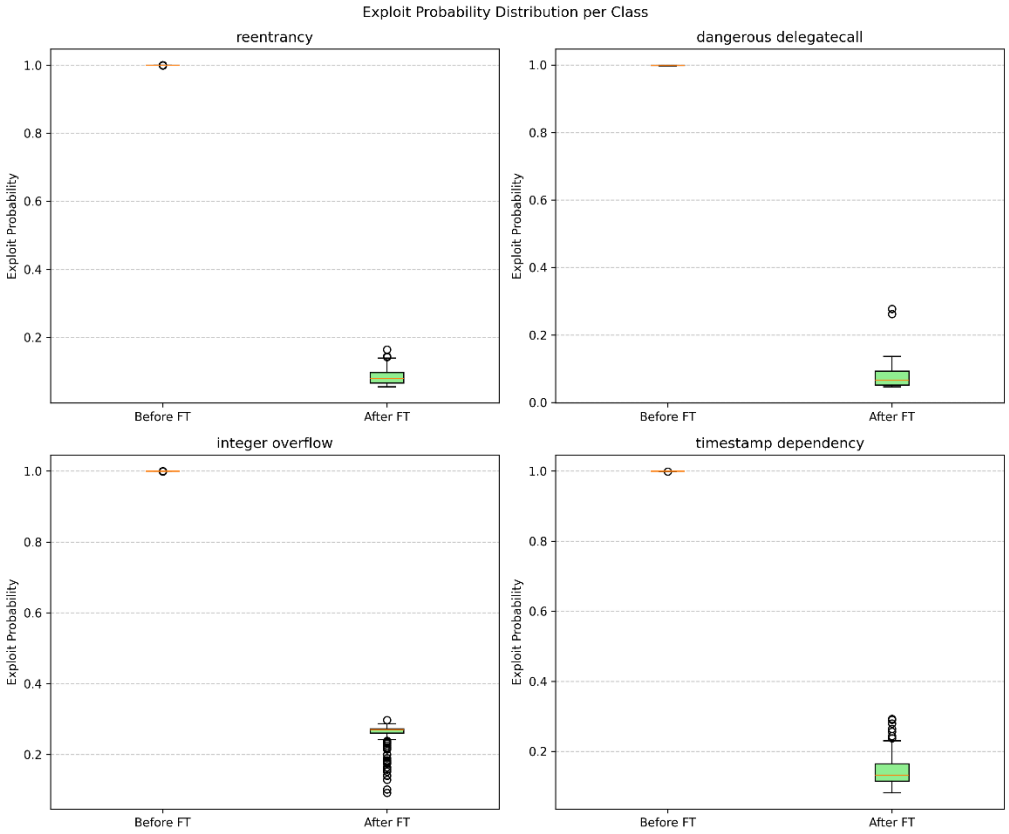

Figure 7. Boxplots of the predicted exploit probability for each class before and after fine-tuning

Each subplot shows the distribution of exploit probability predicted by the exploit head. For each vulnerability class:

- The y-axis is the predicted exploit probability $p_{exp} \in [0,1]$.
- Each boxplot summarizes many test samples of that class.

1) Reentrancy: Before fine-tuning, the median is approximately 1.0, which implies zero variance. After fine-tuning, the median is in the range of 0.07 to 0.08, with a tight interquartile range and a few mild outliers around 0.15. This implies that the exploits of Reentrancy are highly suppressible, and it is possible to reduce the probability of the exploit by making a few changes in causality. So that, the model has learned strong causal levers for reentrancy.
2) Dangerous Delegatecall: The Median is even around 1.0 before fine-tuning. After fine-tuning, the Median ≈ 0.06 with a slightly higher range than reentrancy, though a few values are between 0.25 and 0.30. This test has confirmed that Delegatecall Risk is reducible, though it is more dependent and harder to neutralize with minimal changes. This is

ISSN: 2312-8135 | Print ISSN: 1992-0652 | info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

understandable since Delegatecall is an inherently dangerous vulnerability and, even after fixing it, some risk is always involved.

3) Integer Overflow: Also, the median is about 1.0 before fine tuning. After fine tuning, the median is about 0.25 with wider distribution and several samples are moderately risky. It shows that integer overflow needs more guards.
4) Timestamp Dependency: Also, the median is about 1.0 before fine tuning. After fine tuning Median ≈ 0.13–0.15 with noticeable variance. These results indicate that Timestamp dependency is partially reducible.

The probability distribution graphs for exploit probabilities, as shown in the figure above, indicate a high probability that the model is working well and is effective. The probability distribution graphs for exploit probabilities, as shown in the figure above, indicate a high probability that the model is working well and is effective. The exploit probabilities, prior to fine-tuning, are completely saturated across all vulnerability classes, indicating that the probabilities are mostly around 1.0 with little variation. This is a characteristic of a hard detector rather than a risk estimator. Thus, it is impossible to perform any form of causal intervention and fix generation because changes in the causal variables cannot impact the output. The exploit probabilities, after counterfactual fine-tuning, are significantly reduced and class-dependent. The reentrancy risks indicate the lowest level of post-tuning risks with the lowest spread in probabilities for low probabilities owing to the presence of strong causal levers, while the delegatecall and timestamp risks are moderately high owing to their contextual and semantic characteristics. Finally, the integer overflow risks indicate the highest probabilities with the largest spread in probabilities, thus showing that some contracts are inherently risky even after causal suppression, which is in line with our expectations. It is important to note that none of the classification performance is compromised in the process, as indicated by high macro-F1 and accuracy scores, which point to exploitable controllability without compromising classification quality. The figure above confirms that the exploit head is converted from a saturated classifier to a causal-sensitive risk estimator using counterfactual fine-tuning.

4.4 Analysis of Causal Representations:

The model is able to learn seven causes, and each cause is represented by a probability between 0 and 1. We examine the causes' behavior for the vulnerabilities in three different manners: (i) the average activation of each cause for each vulnerability class, (ii) the correlation of each cause with the TF-IDF features, where we are able to directly map to the code tokens, and (iii) the t-SNE projection of the shared latent vectors, where we are able to visualize the differences between the vulnerability types.

4.4.1 Causal Probabilities per Class:

For each class, we utilized 20 training data instances and computed the model's predicted causal probability in the absence of intervention. Figure 8 displays box plots of these results for each cause and class, and Table 4 summarizes the results in terms of their means and standard deviations.

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

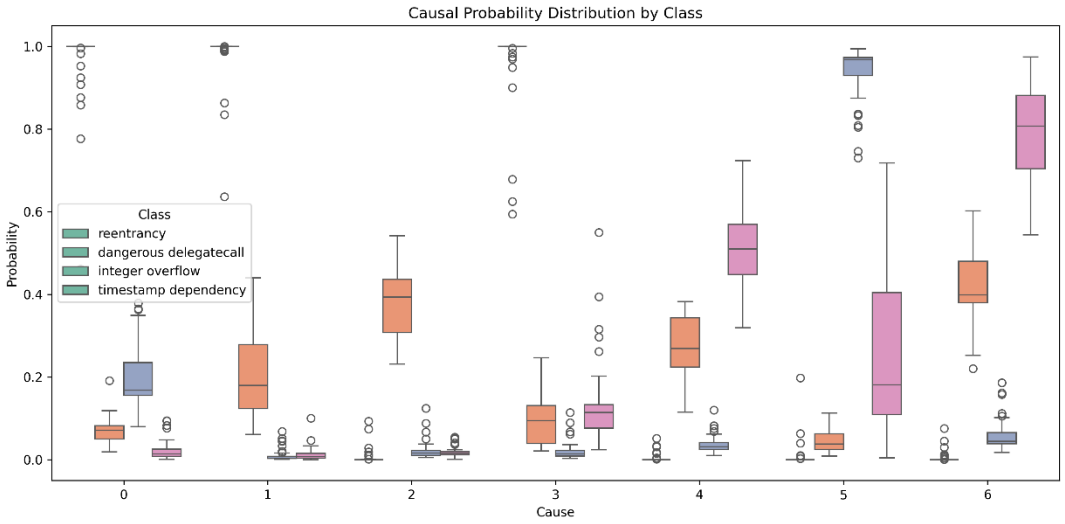

Figure 8. Boxplots of causal probabilities per class

Table 4. Mean causal probabilities (with standard deviation) per class (20 samples each)

Class	Cause 0	Cause 1	Cause 2	Cause 3	Cause 4	Cause 5	Cause 6
reentrancy	0.9502 ± 0.1036	0.9768 ± 0.0582	0.0038 ± 0.0082	0.9492 ± 0.1236	0.0057 ± 0.0151	0.0122 ± 0.0358	0.0077 ± 0.0219
delegatecall	0.0691 ± 0.0387	0.2042 ± 0.1198	0.3858 ± 0.0825	0.0919 ± 0.0616	0.2633 ± 0.0761	0.0513 ± 0.0334	0.4023 ± 0.0787
integer overflow	0.1895 ± 0.0903	0.0119 ± 0.0159	0.0282 ± 0.0286	0.0236 ± 0.0226	0.0490 ± 0.0221	0.9250 ± 0.0692	0.0820 ± 0.0433
timestamp dependency	0.0270 ± 0.0226	0.0102 ± 0.0087	0.0186 ± 0.0146	0.0987 ± 0.0446	0.4758 ± 0.1150	0.3554 ± 0.2070	0.7214 ± 0.1535

These results provide insight into how the model internally attributes vulnerabilities to specific causal factors.

In the case of the reentrancy class, it can be noted that according to the results provided, it can be seen that Cause 0 has extremely high activations at 0.9502, Cause 1 has extremely high activations at 0.9768, and Cause 3 has extremely high activations at 0.9492 with low standard deviations. However, all other causes are nearly zero in value. This indicates that it seems to have learned a pattern in the causes that is extremely consistent, with only a few causes being highly correlated with the vulnerability, and all other causes being nearly zero in value. The low standard deviation also indicates that it has learned this pattern consistently across all samples, possibly due to the well-defined nature of reentrancy vulnerabilities.

In the critical delegatecall class, the characteristics of the causal activations of the critical delegatecall class seem to be even more divergent and diversified. The highest means of the causal activations of the critical delegatecall class are for Cause 6, where the value is 0.4023,

followed by Cause 2, where the value is 0.3858, and then Cause 4, where the value is 0.2633, with moderate standard deviations. Unlike the reentrancy attack, there is no cause that is dominant in all instances, implying that the activation of the delegatecall vulnerabilities depends on a number of factors, a notion that is in line with the observation that the vulnerabilities of the delegatecall depend on the contract architecture, context of calls, and interactions of storage.

For the integer overflow class, it can be observed that the probabilities for all other causes are close to zero, and the model has dominated the probability for Cause 5 (0.9250), which means that the model has learned the single dominant cause for this type of vulnerability effectively. It has also been observed that the standard deviation for this cause is low, which is good for integer overflow vulnerabilities since it is generally caused due to a specific misuse of arithmetic operations.

As far as the timestamp dependency class is concerned, it can be seen that the pattern of causality is again distributed over different causes. The highest level of activation has again been achieved by Cause 6 at 0.7214, followed by Cause 4 at 0.4758 and Cause 5 at 0.3554. Once again, it can be noted that the standard deviations are higher for Cause 5, indicating that there is a higher degree of variation in the samples because it can be caused by different combinations of assumptions in the environment and the conditional statements.

Hence, it can be concluded that the results indicate that the proposed model for causal learning can effectively learn the class-specific causal signature. The patterns of "reentrancy" and "integer overflow" have highly concentrated causal activations, while the patterns of "delegatecall" and "timestamp dependency" have relatively less concentrated causal activations. The difference in the patterns of vulnerabilities is interesting because it indicates that the model is not memorizing the class labels and can effectively learn the class-specific causal signature.

4.4.2 Correlation of Causes with TF-IDF Features:

In order to understand what these causes actually represent in terms of code, we have also calculated the correlation between cause probabilities and TF-IDF feature values over 200 randomly sampled instances from the training data. The top five positive and negative correlations are shown in Table 5. The patterns are highly interpretable and domain-relevant:

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

Table 5. Top five positive and negative correlated features for each cause

Cause	Positive correlations	Negative correlations
0	of (0.756), with (0.661), be (0.629), the (0.627), for (0.625)	public (-0.638), require (-0.549), emit (-0.544), add (-0.412), bool (-0.408)
1	of (0.742), with (0.649), for (0.619), be (0.619), call (0.617)	public (-0.636), require (-0.548), emit (-0.517), returns (-0.421), true (-0.408)
2	let (0.588), assembly (0.534), result (0.439), mload (0.416), ptr (0.352)	for (-0.308), of (-0.289), to (-0.269), with (-0.255), this (-0.247)
3	of (0.757), with (0.663), for (0.646), be (0.632), the (0.628)	public (-0.635), require (-0.556), emit (-0.517), returns (-0.436), true (-0.432)
4	block (0.731), timestamp (0.640), public (0.445), weiamount (0.396), add (0.382)	to (-0.460), of (-0.453), with (-0.393), be (-0.375), the (-0.375)
5	require (0.527), spender (0.496), ownershiptransferred (0.493), true (0.489), allowance (0.475)	of (-0.584), for (-0.521), with (-0.515), this (-0.499), call (-0.494)
6	block (0.691), timestamp (0.597), public (0.458), weiamount (0.419), tokenamount (0.406)	of (-0.465), to (-0.462), with (-0.402), the (-0.385), be (-0.384)

Cause 0 is positively correlated to common English words such as "of," "with," "be," etc., and negatively correlated to keywords such as "public," "require," "emit," which relate to patterns of vulnerabilities in the code written in the Solidity language. This suggests that 'cause 0' is not directly related to the patterns of the vulnerabilities, but to the structure of the code.

Cause 1 shows a similar pattern, correlated with generic tokens and inversely related to typical function rates.

Cause 2 is also closely related to low-level EVM words such as let, assembly, mload, ptr, and revert. These are somewhat common words related to assembly or Yul code. The negative correlations are with common English words, reinforcing that cause 2 detects the presence of assembly.

Cause 3 again mirrors causes 0 and 1, being correlated with generic tokens and anti-correlated with Solidity keywords.

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

Cause 4 is positively correlated with block, timestamp, public, weiamount, add – tokens that appear in timestamp-dependent and arithmetic operations.

Cause 5 is similar to token-related functions such as require, spender, ownershiptransferred, allowance, and balanceof, all of which are Ethereum Request for Comment 20 (ERC20) token and access control functions and all of which are common integer overflow vulnerabilities.

Cause 6 is similar to cause 4 in terms of tokens but also contains weiamount, tokenamount, and state. This again shows a link to timestamp operations and state changes, which suggests that cause 4 and cause 6 could be dealing with different parts of timestamp dependencies, where cause 4 could be seen as a more general cause compared to cause 6.

The heatmap in Figure 9 presents the entire correlation matrix for the top ten tokens of each cause, offering a compact summary of the relationships of each cause to the code features.

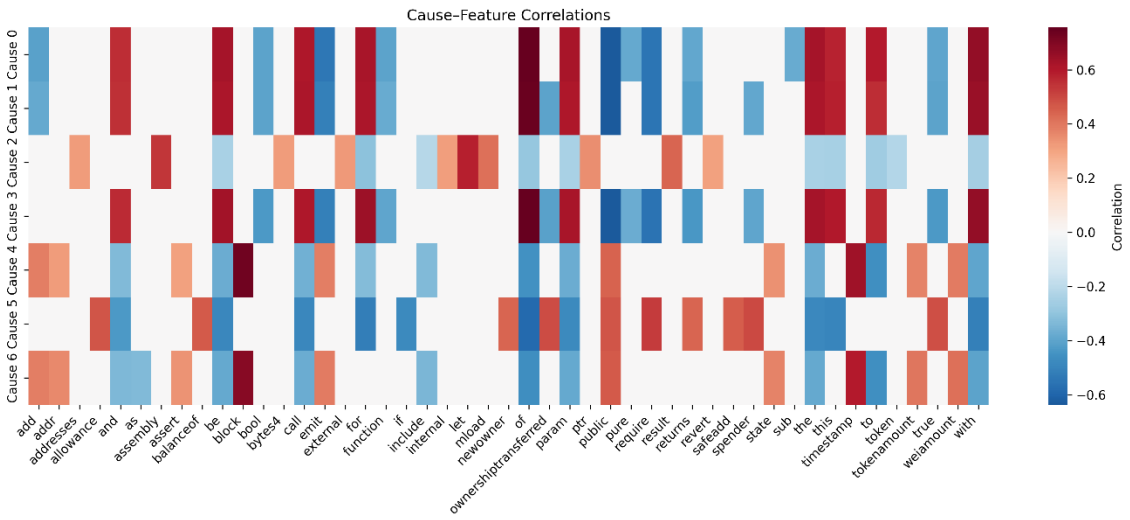

Figure 9. Heatmap of cause–feature correlations

4.4.3 t-distributed Stochastic Neighbor Embedding (t-SNE) Visualizations of Latent Space:

In order to obtain some insights into how it organizes the representations, we visualized the representations that were learned on a random selection of 500 instances in the training data prior to the addition of the causal effect, and we observe from Figure 10 that a 2D visualization of the representations is colored according to the true class of vulnerability.

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

Figure 10. t-SNE projection of latent representations

By referring to the figure above, it is possible to understand the structure of the learned representation by the model before the classification stage. By referring to the figure above, it is possible to see that every point in the figure represents a sample of smart contracts, and the color represents the different classes of vulnerability. The most interesting feature that can be understood from the figure above is that there are different clusters, implying that the model has been able to learn the representation and has separated different classes of vulnerability.

First, the clusters for the different classes appear to be well separated, and there is minimal overlap between the classes. This implies that the encoder has been able to learn discriminative features for the vulnerability types, which is a characteristic of the structural differences. For instance, one of the clusters appears to have formed on the far left of the plot, and another cluster appears to have formed slightly above it. The remaining two clusters appear to have formed in the center of the plot and on the far right, respectively. This implies that contracts with similar vulnerability characteristics have been mapped close to one another, and contracts with different vulnerability mechanisms have been mapped far from one another. This implies the presence of informative semantic features in this representation, which is critical for vulnerability detection.

Second, the fact that the clusters are compact gives us further evidence that the model is stable in the sense that, for each class, the samples are well grouped. This gives us further evidence that the model is well learning the contracts that belong to a given vulnerability class and that are represented in a similar manner. This is important in the analysis of smart contracts because the same vulnerability can be represented in different ways in the code.

Third, the visualization also indicates the differences in the structural complexity of the classes. Some clusters appear very compact and well isolated, while others show slightly elongated or

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

curved shapes. The differences in the patterns are because of the differences in the manifestation of the vulnerabilities in the codes. The well-defined vulnerabilities in the codes resulted in compact clusters, whereas the vulnerabilities that were slightly influenced resulted in the dispersed patterns.

The t-SNE projection has further validated the effectiveness of the proposed causal-aware representation learning method, which has been able to attain a structured and interpretable representation space where different vulnerability classes are located within their own neighborhood. This has further supported the robust classification performance achieved within the previous sections, validating the effectiveness of the model within the context of separating the types of vulnerabilities in an effective manner, which is vital for detection purposes as well as for fix generation.

4.5 Minimal Fix and Interpretability:

A key contribution of this work is the ability to suggest a “minimal fix” – a small modification to the causal vector that reduces the predicted exploit probability below a threshold (0.05) while staying as close as possible to the original vector. The gradient-based optimization of the test sample's gradient to find a causal vector starting from the original prediction aims to find a vector that minimizes the probability of the exploit, where an L1 term is included but is set to zero in the current case to allow movement from the original vector. The optimization is done for up to 2,000 runs using cosine annealing.

4.5.1 Demonstration on Test Samples:

We have tested our minimal fix procedure with the above five test cases. The results obtained with our minimal fix procedure with the above five test cases are given in Table 6, with the exploit probability, best probability found, causal vector, and causes reduced (difference > 0.1). The results obtained with our minimal fix procedure have been successful in reducing the exploit probability to below 0.05 on average with all five test cases and have reduced the exploit probability by 0.15 on average over all five test cases. What is more surprising is that in all five test cases, the causal vector found with our minimal fix procedure is [0, 0, 1, 0, 0, 0, 0], and in all five test cases, only cause 2 is left on, and all other causes are switched off to get the minimum exploit probability for any input. This means that the model has learned a universal representation for safety, in that if cause 2 alone is left on, then it gets the minimum exploit probability for any input.

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

Table 6. Minimal fix results on five test samples

Sample	Original prob	Fixed prob	Reduction	Fixed causal vector	Causes reduced
0	0.1121	0.0272	0.0849	[0,0,1,0,0,0,0]	0,1,3
1	0.2596	0.0422	0.2174	[0,0,1,0,0,0,0]	0,5,6
2	0.2714	0.0421	0.2293	[0,0,1,0,0,0,0]	0,5
3	0.0650	0.0152	0.0498	[0,0,1,0,0,0,0]	0,1,3
4	0.0636	0.0149	0.0487	[0,0,1,0,0,0,0]	0,1,3

The causes that are reduced vary for each sample depending on the original causal profile. Consider sample 0, 3, 4. In this sample, the causes have high activation for causes 0, 1, 3. This is expected for the reentrancy bug, so all of these causes were reduced to fix sample 0, 3, 4. In sample 1, the causes have moderate activation for causes 0, 5, 6. This is expected for the timestamp/integer overflow bug, so all of these causes were reduced to fix sample 1. In all of these cases, cause 2 remains active.

4.5.2 Cause Reduction Frequency:

In figure 11, we can observe how often a cause (Cause0, Cause1,... , and Cause6) is reduced in the minimal fix search optimization process for the different classes of vulnerability. This is, we can observe how often the optimization process has chosen that the best way to reduce the probability of exploit is by reducing a cause variable. This provides insight into which latent causes are most responsible for different types of smart-contract vulnerabilities.

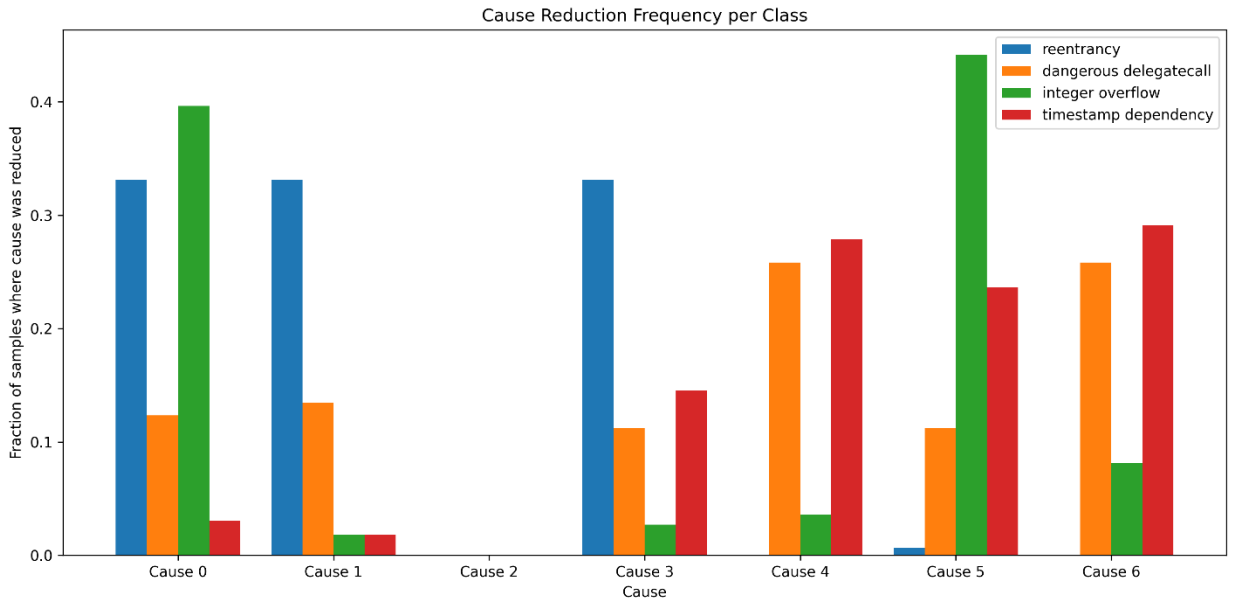

Figure 11. Frequency of cause reduction per class

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

For the reentrancy class, the results show that Cause 0, Cause 1, and Cause 3 are the most frequently reduced factors, each appearing in roughly one-third of the fixes. This means that the model has been identifying reentrancy vulnerabilities and linking them to specific dominant causal patterns. However, the rest of the causes have been rarely modified, suggesting their lack of importance in reentrancy exploits.

For delegatecall, the frequencies are more spread out. Cause 4 and Cause 6 are the most frequently reduced (around 0.25 each), followed by moderate reductions in Causes 0, 1, 3, and 5. This distribution suggests that delegatecall vulnerabilities arise from multiple interacting factors rather than a single dominant cause. The model is therefore investigating a range of adjustments in the causes when it is seeking a minimal fix.

In the integer overflow class, Cause 0 and Cause 5 dominate strongly, with Cause 5 showing the highest reduction frequency overall (around 0.44). This means the optimization procedure frequently reduces this factor to eliminate the overflow risk, indicating that it represents a key latent pattern related to arithmetic operations or unsafe numeric handling. The other causes are relatively rare and support the view that integer overflow is dominated by a few causes.

For timestamp dependency, the pattern is again more distributed. Cause 6 and Cause 4 are the most frequently reduced, followed by moderate contributions from Causes 3 and 5. This suggests that timestamp vulnerabilities are associated with several related causal patterns, likely reflecting different ways developers rely on block.timestamp or time-dependent logic.

5. CONCLUSIONS

In this study, we introduced a new causal-aware deep learning approach for smart contract vulnerability analysis, which is not possible with the traditional approaches for vulnerability detection using the causal factors of the vulnerabilities and providing the minimum corrective actions for the detected vulnerabilities. The proposed approach is based on the use of a multi-task neural model with three tasks, namely vulnerability classification, exploit probability estimation, and causal attribution. We used TF-IDF embeddings for the smart contract code and a shared neural model for learning the underlying representation of the smart contracts, enabling the model to perform the tasks of classification and causal attribution. The results of the proposed approach were tested using the SC_4label dataset of 2217 smart contracts, and impressive results were obtained in the experiment using the proposed approach. The proposed approach obtained a macro F1 of 0.899 and overall accuracy of 0.95 in the experiment. The researchers observed the best detection rates for the vulnerabilities of reentrancy (F1=0.99), integer overflow (F1=0.93), and reasonable detection rates for the less common type of vulnerability, i.e., delegatecall (F1=0.77). The causal analysis also revealed the pattern between the types of vulnerabilities and the causal factors. For example, the reentrancy contracts indicated high probabilities for Cause 0, Cause 1, and Cause 3, with probabilities of 0.9502 ± 0.1036, 0.9768 ± 0.0582, and 0.9492 ± 0.1236, respectively. The integer overflow contracts also indicated high correlation with Cause 5 at a probability of 0.9250 ± 0.0692. The proposed minimal fix optimization approach also proved the feasibility of the framework in suggesting the fixes. As illustrated, the proposed framework

reduced the probability of the exploit significantly for the test samples. For example, the proposed framework reduced the probability from 0.2596 to 0.0422, and the second instance reduced the probability from 0.1121 to 0.0272. The main difference of the proposed approach, with regard to other conventional vulnerability detectors, is that, apart from the detection of insecure code, the cause to fix paradigm is incorporated. Therefore, the main novelty of the proposed approach is the introduction of the cause to fix paradigm, which is the main difference with other conventional detectors. There are also some possible ways to further improve the above-proposed approach. Firstly, it is possible to improve the above-proposed approach related to improving the causal modeling module by using graph-based program models, such as abstract syntax trees and control flow graphs, for representing smart contracts and modeling causal dependencies. Secondly, it is possible to improve the above-proposed approach by using more and diverse data, except for the above four types of vulnerabilities. Thirdly, it is possible to improve the above-proposed approach by using large language models for code repair, and the minimal causal changes can be directly translated into code patches. There are several significant benefits that could be provided by conducting the suggested research to the domain of smart contract security, which can include both theoretical and practical advantages. Firstly, theoretically, the fact can be proven that the use of causal learning together with multi-task learning technologies is applicable to solving problems related to security; in other words, moving away from correlation analysis of a smart contract vulnerability will help identify not only a vulnerability but also its root cause. Furthermore, considering a practical standpoint, the proposed method provides a sound basis for the implementation of automatic mitigation of vulnerabilities. In light of the use of counterfactual methods, it is possible to propose minimal causes that will lead to a noticeable decline in the probability of an exploit. As a consequence, this means that it is feasible to develop auditing technology for developers to address their vulnerabilities. Finally, considering different factors according to the nature of the vulnerability (e.g., reentrancy or integer overflow) simplifies the understanding of the results produced by the model. This increases the possibility that such an AI model will be accepted and implemented in vital fields like the blockchain industry.

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

مجلة جامعة بابل للعلوم الصرفة والتطبيقية مجلة جامعة بابل للعلوم الصرفة والتطبيقية

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

Conflict of interests.

There are non-conflicts of interest.

References

[1] A. Jumagaliyeva, A. Tulegulov, G. Murzabekova, and G. Muratova, "Application of smart contracts in electronic systems based on blockchain technologies," *kazutb,* vol. 2, no. 23, 2024.

[2] R. Kaur, A. Ali, and M. Faisal, "Smart contracts: the self-executing contracts," in *Blockchain*: Chapman and Hall/CRC, 2022, pp. 35-49.

[3] C. Sendner *et al.*, "Smarter Contracts: Detecting Vulnerabilities in Smart Contracts with Deep Transfer Learning," in *NDSS*, 2023.

[4] S. Al-E'mari and Y. Sanjalawe, "A Review of Reentrancy Attack in Ethereum Smart Contracts," in *International Conference on Computing and Communication Networks*, 2023: Springer, pp. 53-70.

[5] Q. Huang, Z. Zeng, and Y. Shang, "An empirical study of integer overflow detection and false positive analysis in smart contracts," in *Proceedings of the 2024 8th International Conference on Big Data and Internet of Things*, 2024, pp. 247-251.

[6] D. He, R. Wu, X. Li, S. Chan, and M. Guizani, "Detection of vulnerabilities of blockchain smart contracts," *IEEE Internet of Things Journal,* vol. 10, no. 14, pp. 12178-12185, 2023.

[7] S. A. Amri, L. Aniello, and V. Sassone, "A review of upgradeable smart contract patterns based on openzeppelin technique," *The Journal of The British Blockchain Association,* 2023.

[8] A. Ghaleb, "Towards effective static analysis approaches for security vulnerabilities in smart contracts," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1-5.

[9] P. Gong, W. Yang, L. Wang, F. Wei, K. HaiLaTi, and Y. Liao, "Gratdet: Smart contract vulnerability detector based on graph representation and transformer," *Computers, Materials, & Continua,* vol. 76, no. 2, p. 1439, 2023.

[10] P. Chatterjee, "Smart contracts and machine learning: exploring blockchain and AI in Fintech," *Indian J. Sci. Technol,* vol. 18, no. 2, pp. 113-124, 2025.

[11] L. Brent, N. Grech, S. Lagouvardos, B. Scholz, and Y. Smaragdakis, "Ethainter: a smart contract security analyzer for composite vulnerabilities," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 454-469.

[12] N. Lu, B. Wang, Y. Zhang, W. Shi, and C. Esposito, "NeuCheck: A more practical Ethereum smart contract security analysis tool," *Software: Practice and Experience,* vol. 51, no. 10, pp. 2065-2084, 2021.

[13] A. Ali, Z. U. Abideen, and K. Ullah, "SESCon: Secure Ethereum smart contracts by vulnerable patterns' detection," *Security and Communication Networks,* vol. 2021, no. 1, p. 2897565, 2021.

[14] N. F. Samreen and M. H. Alalfi, "Smartscan: an approach to detect denial of service vulnerability in ethereum smart contracts," in *2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, 2021: IEEE, pp. 17-26.

[15] Y. Liu, Y. Li, S.-W. Lin, and Q. Yan, "ModCon: A model-based testing platform for smart contracts," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1601-1605.

[16] M. Ashouri, "Etherolic: a practical security analyzer for smart contracts," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 353-356.

[17] T. D. Nguyen, L. H. Pham, J. Sun, Y. Lin, and Q. T. Minh, "sfuzz: An efficient adaptive fuzzer for solidity smart contracts," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 778-788.

[18] T. Chen *et al.*, "SODA: A Generic Online Detection Framework for Smart Contracts," in *NDSS*, 2020.
[19] 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.
[20] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "Smartbugs: A framework to analyze solidity smart contracts," in *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, 2020, pp. 1349-1352.
[21] G. A. Pierro, "Smart-graph: Graphical representations for smart contract on the ethereum blockchain," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021: IEEE, pp. 708-714.
[22] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural networks," in *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, 2021, pp. 3283-3290.
[23] Z. Zhen, X. Zhao, J. Zhang, Y. Wang, and H. Chen, "DA-GNN: A smart contract vulnerability detection method based on Dual Attention Graph Neural Network," *Computer Networks,* vol. 242, p. 110238, 2024.
[24] 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.
[25] S.-J. Hwang, S.-H. Choi, J. Shin, and Y.-H. Choi, "CodeNet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection," *IEEE Access,* vol. 10, pp. 32595-32607, 2022.
[26] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, "Eth2vec: learning contract-wide code representations for vulnerability detection on ethereum smart contracts," in *Proceedings of the 3rd ACM international symposium on blockchain and secure critical infrastructure*, 2021, pp. 47-59.
[27] J. Huang, K. Zhou, A. Xiong, and D. Li, "Smart contract vulnerability detection model based on multi-task learning," *Sensors,* vol. 22, no. 5, p. 1829, 2022.
[28] 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," *IEEE Access,* vol. 12, pp. 23549-23567, 2024.
[29] Q. Qian, Y. Qin, J. Luo, Y. Wang, and F. Wu, "Deep discriminative transfer learning network for cross-machine fault diagnosis," *Mechanical systems and signal processing,* vol. 186, p. 109884, 2023.
[30] J. Stiensmeier-Pelster and H. Heckhausen, "Causal attribution of behavior and achievement," in *Motivation and action*: Springer, 2025, pp. 693-744.
[31] W. R. Monteiro and G. Reynoso-Meza, "Counterfactual generation through multi-objective constrained optimisation," 2022.
[32] S. Qaiser and R. Ali, "Text mining: use of TF-IDF to examine the relevance of words to documents," *International journal of computer applications,* vol. 181, no. 1, pp. 25-29, 2018.
[33] H. Chu, P. Zhang, H. Dong, Y. Xiao, and S. Ji, "Deepfusion: Smart contract vulnerability detection via deep learning and data fusion," *IEEE Transactions on Reliability,* vol. 74, no. 3, pp. 3544-3558, 2024.
[34] R. Kruse, S. Mostaghim, C. Borgelt, C. Braune, and M. Steinbrecher, "Multi-layer perceptrons," in *Computational intelligence: a methodological introduction*: Springer, 2022, pp. 53-124.
[35] C. Chen and L. Fan, "Interpretability of statistical, machine learning, and deep learning models for landslide susceptibility mapping in Three Gorges reservoir area," *arXiv preprint arXiv:2405.11762,* 2024.

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

[36] M. Zidan, M. G. Eldin, M. Y. Shams, M. Tolan, A. Abd-Elhamed, and M. Abdel-Aty, "A quantum algorithm for evaluating the hamming distance," *Computers, Materials & Continua,* vol. 71, no. 1, 2022.

[37] F. Shen, J. Sun, X. Xie, and L. Zhang, "Research on Road Condition Image Conversion in the Day and Night Domain Based on the CycleGAN Method," in *Proceedings of the 2023 7th International Conference on Electronic Information Technology and Computer Engineering*, 2023, pp. 303-311.

[38] D. Ramos, J. Franco-Pedroso, A. Lozano-Diez, and J. Gonzalez-Rodriguez, "Deconstructing cross-entropy for probabilistic binary classifiers," *Entropy,* vol. 20, no. 3, p. 208, 2018.

[39] M. Tao, "Minimization of L_1 Over L_2 for Sparse Signal Recovery with Convergence Guarantee," *SIAM Journal on Scientific Computing,* vol. 44, no. 2, pp. A770-A797, 2022.

[40] A. R. Ismail, M. Z. R. Azhary, and N. A. Hitam, "Evaluating adan vs. adam: an analysis of optimizer performance in deep learning," in *International Symposium on Intelligent Computing Systems*, 2024: Springer, pp. 251-263.

[41] M. Zhang, W. Li, L. Zhang, H. Jin, Y. Mu, and L. Wang, "A Pearson correlation-based adaptive variable grouping method for large-scale multi-objective optimization," *Information Sciences,* vol. 639, p. 118737, 2023.

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

مجلة جامعة بابل للعلوم الصرفة والتطبيقية مجلة جامعة بابل للعلوم الصرفة والتطبيقية

الخلاصة

أُفيد بأن العقود الذكية في تقنية البلوك تشين أكثر عرضةً للثغرات الأمنية الناتجة عن البرمجة. وقد أشارت التقارير إلى أن التقنيات التقليدية للكشف عن الثغرات تركز بشكل أساسي على تحديدها، دون التطرق إلى أسبابها والإجراءات المتخذة حيالها. تقترح هذه الورقة البحثية إطار عمل قائم على مفهوم التعلم العميق الواعي بالعلاقة السببية، حيث يتم فيه دمج تصنيف الثغرات، والتنبؤ باحتمالية استغلالها، وتحديد أسبابها، وذلك بالنسبة لثغرات العقود الذكية. وقد تم دمج تقنيات تعلم التمثيل الكامن لرمز العقد الذكي باستخدام تضمينات تردد المصطلح – تردد المستند العكسي (TF–IDF) وشبكة عصبية مشتركة في النموذج المقترح. في التجربة، تألفت مجموعة البيانات من 2217 عقدًا ذكيًا، تم تصنيفها إلى أربع فئات من الثغرات. شملت هذه الفئات إعادة الدخول، واستدعاء المندوب الخطير، وتجاوز سعة العدد الصحيح، والاعتماد على الطابع الزمني. وقد لوحظ أن النموذج حقق دقة إجمالية بلغت 0.95، ودرجة F1 الكلية بلغت 0.899. بلغت قيم F1 0.99 لإعادة الدخول، و0.93 لتجاوز سعة الأعداد الصحيحة، و0.91 لاعتماد الطابع الزمني، و0.77 لاستدعاء المندوب. يعكس الأداء المنخفض نسبيًا لاستدعاء المندوب التعقيد المتأصل في هذا النوع من الثغرات الأمنية، والذي يعتمد على سياق التنفيذ الخارجي، ومحاذاة التخزين، والتفاعلات بين العقود. من ناحية أخرى، يُعتبر انخفاض أداء هذا النوع من الثغرات الأمنية طبيعيًا نظرًا لعدم توازن مجموعة البيانات، التي تحتوي على 97 عقدة فقط خاصة بهذه الثغرة. أظهر التحليل السببي للتجربة علاقات فريدة بين الثغرات الأمنية والعوامل السببية المُستنتجة. قدمت التجربة وحدة تحسين الإصلاح الأدنى لتوليد تعديلات سببية مضادة للواقع لتقليل مخاطر الاستغلال مع الحفاظ على التنبؤات التي قدمها النموذج. أظهرت التجربة انخفاضًا ملحوظًا في احتمالية الاستغلال، على سبيل المثال، من 0.2596 إلى 0.0422 ومن 0.1121 إلى 0.0272 لعينات تمثيلية. كما تُظهر نتائج التحليل العرضي أن ثغرات استدعاء المندوب ترتبط بعوامل سببية متعددة ذات تفعيل متوسط، بدلاً من سبب رئيسي واحد، مما يزيد من صعوبة التصنيف. وهذا يُوضح صعوبة اكتشاف هذه الثغرة، ويُبرر سببًا آخر لانخفاض أداء تصنيفها. وعلى عكس النظام التقليدي، تُقدم طريقتنا نموذجًا جديدًا لـ"سبب الإصلاح" لتحليل نقاط الضعف.

المقدمة:

تزداد العقود الذكية في أنظمة البلوك تشين عرضةً للثغرات الأمنية الناتجة عن أخطاء البرمجة. وتركز تقنيات الكشف عن الثغرات التقليدية بشكل أساسي على تحديدها دون تحليل أسبابها الجذرية أو تقديم حلول عملية. هذا القصور يحفز تطوير مناهج قادرة على كشف الثغرات وتفسير أسبابها.

طرق العمل:

تقترح هذه الدراسة إطار عمل للتعلم العميق يراعي العلاقات السببية لتحليل ثغرات العقود الذكية، حيث يدمج تصنيف الثغرات، والتنبؤ باحتمالية استغلالها، وتحديد أسبابها. يتعلم النموذج تمثيلات كامنة لرمز العقد الذكي باستخدام تضمينات تردد المصطلح – تردد المستند العكسي (TF–IDF) مع بنية شبكة عصبية مشتركة. تتكون مجموعة البيانات التجريبية من 2217 عقدًا ذكيًا مصنفة إلى أربع فئات من الثغرات: إعادة الدخول، واستدعاء المندوب الخطير، وتجاوز سعة عدد صحيح، والاعتماد على الطابع الزمني.

الاستنتاجات:

يُوسّع الإطار المقترح نطاق الكشف التقليدي عن الثغرات الأمنية من خلال إدخال الاستدلال السببي والحلول البديلة البسيطة. لا يقتصر هذا النهج على تحديد الثغرات الأمنية فحسب، بل يشرح أيضًا أسبابها ويقترح تعديلات مُوجّهة، مما يُقدّم نموذجًا جديدًا لـ"سبب الإصلاح" لتحسين أمان العقود الذكية.

الكلمات المفتاحية:

أمن العقود الذكية، اكتشاف الثغرات الأمنية، التعلم العميق السببي، التعلم متعدد المهام، التحليل المضاد للواقع، الإصلاح الآلي للثغرات الأمنية، أمن سلسلة الكتل (البلوك تشين).