ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 7, July 2023

Code clone detection in Smart Contract: A Survey

[1] Amandeep kaur, [2]Gurpreet singh, [3]Himanshu aggarwal

[1][2][3] Department of Computer Science and Engineering, Punjabi University Patiala, India
[1] amandeepdhiman45@gmail.com, [2]Gurpreet.1887@gmail.com, [3]himanshu@pbi.ac.in

Abstract*— Blockchain technology has experienced exponential growth in the past few years and has provided a secure and decentralized infrastructure for various applications, including smart contracts. Smart contracts have revolutionized the way financial and business transactions are conducted by eliminating the need for intermediaries and enabling transparency, automation, and immutability. However, the increasing complexity of smart contracts and the lack of proper testing and verification have resulted in numerous bugs and vulnerabilities, leading to significant financial losses for businesses and individuals. Code clone detection has emerged as an effective approach to ensuring the reliability and security of smart contracts by identifying and addressing potential vulnerabilities. This paper presents a survey of the current status of smart contract clone detection. We address several research questions, including the methods used for smart contract clone detection, the available tools, and the machine learning and deep learning techniques used for this purpose. The study highlights the importance of clone detection in smart contracts and presents a comprehensive analysis of existing approaches for detecting clones. By understanding the state of the art in this area, researchers and practitioners can identify gaps in current approaches and develop new and more effective methods for detecting code clones in smart contracts, thus ensuring their security and reliability.

***Index Terms*— Code Clone, Smart contract, code reuse, Blockchain, Ethereum.**

I. INTRODUCTION

The technology behind blockchain enables a network of computers to collaborate and administer a chronological database of transaction records that is shared by all of the computers in the network. As a result of the system's decentralized design, transactions do not require the participation of a centralized authority, such as a bank or credit card firm. Accountability and openness are both improved by the fact that every transaction is kept in a public ledger that any user on the network may access and examine. On the blockchain, Ethereum is a decentralized platform for the construction of smart contracts. On a blockchain platform like as Ethereum, the code for a transaction is stored and kept indefinitely; the network then "runs" this code to complete the transaction. Etherscan and Openzeppelin have recently released over a million free and open-source smart contracts. Anyone who uses Ethereum can create and deploy their own smart contracts. Several developers are now making their source code public in order to ensure their customers that their contracts are safe and secure. As smart contracts become increasingly common, various research on the code used to generate them have been undertaken. Code cloning is the step of copying and pasting the code without modification, leading to the reuse of code fragments in different parts of a program. In the context of smart contracts, code cloning can pose even more severe consequences, as vulnerabilities or errors introduced in one contract can easily propagate to other contracts that reuse the cloned code. Furthermore, since smart contracts are immutable once deployed, any vulnerabilities or errors introduced through code cloning cannot be easily fixed or patched. As such, it is essential to prevent or minimize code cloning in smart contract development.

Our primary focus is on investigating the cloned code found in smart contracts. The code within these contracts must be written correctly to ensure their proper functioning; however, this task can prove challenging due to its complexity and the high risk of introducing errors or vulnerabilities into the system. Clones discovered early in the development process allow teams to resolve them before they pose big issues. According to the study report that was quoted [12], the dataset contained 10 million smart contracts that were put into use between July 2015 and December 2018. They noticed that many copies of contracts contained security weaknesses that existed in the originals.

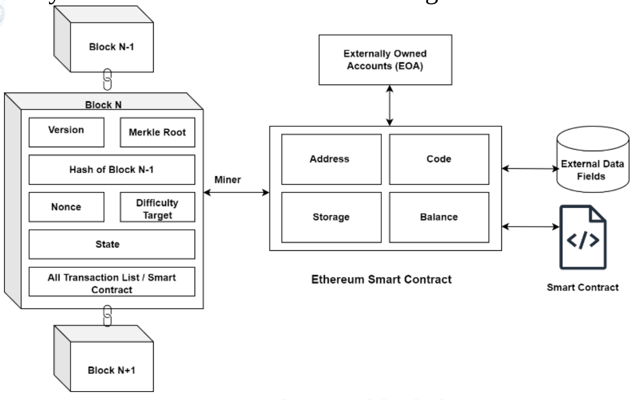


Figure 1: Ethereum blockchain

The smart contract has various real-world applications. Some application examples are as follows:

Healthcare: Smart contracts can be used to automate healthcare procedures such as patient record administration, insurance claim processing, and clinical trial data management. Health systems can use contracts to reduce costs, improve data security, and improve productivity.

Decentralized finance (DeFi): On top of smart contract systems like Ethereum, decentralized finance (DeFi) is a developing field of financial applications. DeFi protocols enable a wide variety of financial operations to take place without the need for a bank or an agent, including lending, borrowing, trading, insurance, and financial transactions.

Intellectual property: Smart contracts can help automate the management of intellectual property rights, including patents, copyrights, and trademarks. Smart contracts can automate the management of intellectual property rights, ensuring creators and owners receive payments and their rights are protected.

Supply chain management: Supply chain management can be performed on smart contract systems such as Ethereum to increase transparency and efficiency. Smart contracts can automate supply chain processes, enforce agreements, and ensure compliance. Supply chain management with blockchain technology can reduce fraud and errors while also providing a tamper-proof and traceable record of transactions.

Banking: Smart contract systems can be used to execute banking, making financial transactions faster and more effective. Smart contracts can automate banking processes such as account administration and payments while ensuring compliance with regulatory requirements. The transparency and security of blockchain-based banking can improve confidence while reducing costs.

The goal of this research is to conduct a detailed review of the available literature on finding code clones in smart contracts. This review aims to cover the various techniques, tools, and approaches used for detecting code clones in smart contracts, and to identify the challenges and issues associated with code clone detection in this context.

The study will explore the following research questions:

RQ1 : What is the current status of Smart contract clone detection?

RQ1.1:What methods are used for smart contract clone detection?

RQ2.What tools are available for smart contract clone detection?

RQ3.Which Machine Learning and deep learning techniques used for smart contract clone detection ?

II. BACKGROUD

2.1 BLOCKCHAIN

A blockchain is a transactional database that is decentralized, public, and chronological. This database is contributed to and updated by all nodes in a P2P network. There are currently several blockchain platforms to choose from, including Bitcoin and Ethereum [16]. A wide variety of applications, such as voting systems, identity verification, and management of supply chains, are currently exploring the potential of blockchain technology. The term "blockchain" refers to the process of recording transactions in this context. Transactions are divided into blocks, which are linked together in a chain. A hash that is exclusive to that block is generated by taking into account the data in the previous block as well as the data in the block that is currently being processed.

Blockchain technology has the potential to boost productivity and transparency across a wide range of businesses. A blockchain, for example, can be used in SCM to record and verify the movement of products, as well as their legitimacy. It can be linked with voting systems to provide a transparent and unchangeable log of votes, increasing public trust in the integrity of elections.

2.2 ETHEREUM

Ethereum is a distributed ledger platform that was first proposed in 2014 and has since grown to become the most widely used blockchain platform that enables smart contracts. Ethereum is open source. Ether, sometimes known as ETH, is the native cryptocurrency of the prominent blockchain platform Ethereum, which was introduced in July 2015 after the network had already gained significant traction [16]. Ethereum separates between contract accounts, also known as simply contracts, and externally owned accounts (EOA), usually referred to as users [17]. Ethereum is built on accounts, which may be managed by either code or a public/private key pair (referred to as external accounts; these are for users) (called contract accounts). Storage (basically a random-access memory that converts 256-bit addresses to 256-bit values) and an ether credit balance are shared by both sorts of accounts (the unit of currency in Ethereum). Ethereum also provides a storage system that enables smart contracts to store and retrieve data. This storage system is designed to be permanent and tamper-proof, ensuring that data stored on the Ethereum network is secure and can be accessed by anyone who has the necessary permissions.

2.3 SMART CONTRACT

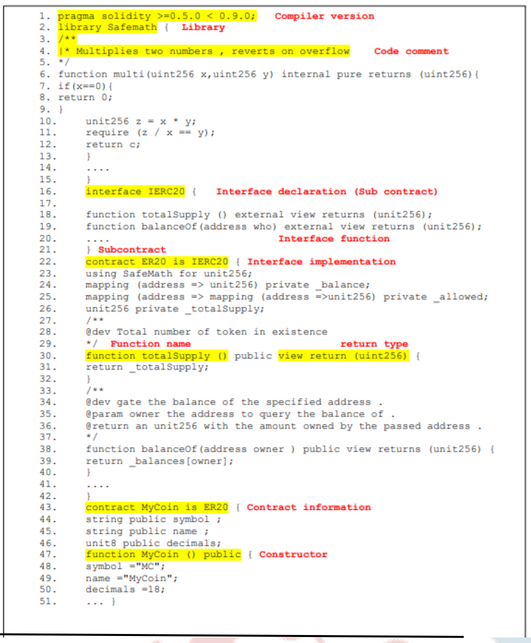
The idea of a smart contract was initially presented for the first time in 1994 by Nick Szabo [8]. An adaptable piece of software that is capable of performing any task is known as a smart contract. When it comes to the creation of Ethereum smart contracts, Solidity is a popular choice. The syntax of the object-oriented language Solidity is remarkably similar to that of the programming language Java. Subcontracts, interfaces, and libraries are the various subcomponents that make up a Solidity smart contract. The term "code blocks" refers to these three different kinds of building blocks [9]. Once they have been added to the blockchain, smart contracts, which are built on the principle that the underlying blockchain technology is immutable, cannot be modified in any way. When a contract has been fulfilled in its entirety, all subsequent operations are governed by the corresponding code. Nobody, not even the person who made it, can make any changes to it. Smart contracts cannot be executed without

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 7, July 2023

the Ethereum Virtual Machine (EVM) [14]. A library is a collection of functions that may be used repeatedly without having to rewrite the code each time the situation or amount of processing time changes. The library, like the Java interface, has no state and no state variables. In Solidity, an interface is an abstract contract with no implemented functionality. A subcontract is a legal agreement that allows ideas to be put into action. State variables can be accessed directly as well as through state variable updating and updating methods. A subcontract in Solidity often inherits and uses an interface.



```
pragma solidity >=0.5.0 < 0.9.0;    Compiler version
library Safemath {  Library
/**
|* Multiplies two numbers , reverts on overflow    Code comment
*/
function multi(uint256 x,uint256 y) internal pure returns (uint256){
if(x==0){
return 0;
}
    unit256 z = x * y;
    require (z / x == y);
    return c;
    }
    ....
    }
    interface IERC20 {    Interface declaration (Sub contract)

    function totalSupply () external view returns (unit256);
    function balanceOf(address who) external view returns (unit256);
    ....                                      Interface function
    } Subcontract
    contract ER20 is IERC20 { Interface implementation
    using SafeMath for unit256;
    mapping (address => unit256) private _balance;
    mapping (address => mapping (address =>unit256) private _allowed;
    unit256 private _totalSupply;
    /**
    @dev Total number of token in existence
    */  Function name                          return type
    function totalSupply () public view return (uint256) {
    return _totalSupply;
    }
    /**
    @dev gate the balance of the specified address .
    @param owner the address to query the balance of .
    @return an unit256 with the amount owned by the passed address .
    */
    function balanceOf(address owner ) public view returns (unit256) {
    return _balances[owner];
    }
    ....
    }
    contract MyCoin is ER20 { Contract information
    string public symbol ;
    string public name ;
    unit8 public decimals;
    function MyCoin () public { Constructor
    symbol ="MC";
    name ="MyCoin";
    decimals =18;
    ... }
```

Figure 2: An example of a smart contract

2.4 CODE CLONE

The act of copying and pasting a section of source code into another piece of code, with or without significant changes, is known as "code cloning." The method of "Code cloning" and the cloned code are both known as "Code cloning." [1][2].

Type of Code clone

(Exact clones **)Type 1:** Code segments that are similar except modifications in comments, layouts, and white space.

(Renamed clones**)Type 2:** Code snippets that are similar save for small formatting, name, or commenting changes. These are also referred to as parameterized copies.

(Near Miss clones)**Type 3**: code segments that have been duplicated and modified, such as by adding or removing statements and changing , identifiers, literals, types, and layouts. These clones are also referred to as gapped clones.

(Semantic clones)**Type 4:** Code segments that are similar in terms of functionality but implemented with different syntactic variations .

2.5 WHY CLONE IN SMART CONTRACT

Clones in smart contracts refer to the duplication of code blocks within different contracts, which can pose significant security risks. Here are some reasons why clone detection is important in smart contracts:

Security risks: Cloned code blocks can contain vulnerabilities or bugs, and if one contract is compromised, all clones can be compromised as well. Identifying and eliminating clones can reduce the chances of a security breach.

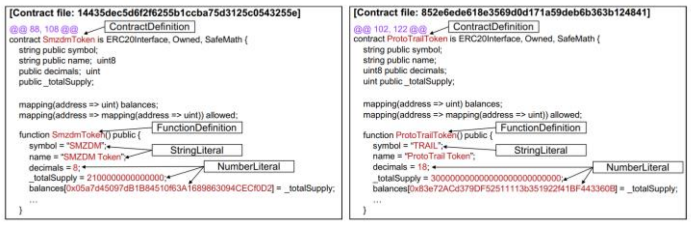
Code maintenance: When the same code block is used in multiple contracts, it can become challenging to maintain and update the code. Clone detection can help identify these instances and facilitate code maintenance.

Efficiency: Duplication of code blocks in different contracts can lead to inefficiencies in storage and processing power, especially in the case of blockchain-based systems with limited resources. Identifying and eliminating clones can help optimize resource usage.

Legal compliance: Cloning code can result in copyright infringement and legal liabilities. Clone detection can help developers ensure that their contracts do not violate any legal or intellectual property rights.

2.5.1 HOW CODE REUSE WITH RENAMING IN SMART CONTRACT

In smart contract development, renaming is a common technique used for code reuse. This technique involves copying existing code segments and renaming the contract definition, function definition, string literals, and number literals. We can see an example of code reuse in a smart contract depicted in Figure 3. This particular example includes renaming multiple nodes of the same type but with different values in order to produce a Type 2 clone.



```
[Contract file: 14435dec5d6f2f6255b1ccba75d3125c0543255e]
@@ 88, 108 @@
contract SmzdmToken is ERC20Interface, Owned, SafeMath {
  string public symbol;
  string public name;  uint8
  public decimals;  uint
  public _totalSupply;

  mapping(address => uint) balances;
  mapping(address => mapping(address => uint)) allowed;

  function SmzdmToken() public {
    symbol = "SMZDM";
    name = "SMZDM Token";
    decimals = 8;
    _totalSupply = 2100000000000000;
    balances[0x05a7d45097dB1B84510f63A1689863094CECf0D2] = _totalSupply;
    ...
  }
```

```
[Contract file: 852e6ede618e3569d0d171a59deb6b363b124841]
@@ 102, 122 @@
contract ProtoTrailToken is ERC20Interface, Owned, SafeMath {
  string public symbol;
  string public name;
  uint8 public decimals;
  uint public _totalSupply;

  mapping(address => uint) balances;
  mapping(address => mapping(address => uint)) allowed;

  function ProtoTrailToken() public {
    symbol = "TRAIL";
    name = "ProtoTrail Token";
    decimals = 18;
    _totalSupply = 3000000000000000000000000000;
    balances[0x83e72ACd379DF52511113b351922f41BF443360B] = _totalSupply;
    ...
  }
```

Figure 3: In this example [9] Type 2 clone in smart contract. Rename the contract definition, function definition, string literal and Number literal. The red strings each represent a separate node that is of the same type but has a different value. The beginning and ending lines of each code example are indicated in purple by the strings.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 7, July 2023

III. 3. SELECTION PROCEDURE

3.1 Selection of primary study

The following keywords have been selected for primary study
searches:
"Blockchain" AND "Smart contract" AND "Code Cloe " AND "Ethereum and "Code reuse "
Google Scholar ,IEEE , Springer and Science Direct this platforms are used in Primary Studies .
For each paper found with paper title , Abstract , Conclusion and Full Text .The remainder of each paper was then examined to identify significant findings and remaining issues.

Sr. no	Sources	E-Sources	Type	Search string	Year
1	IEEE	ieeexpolre.ieee.org	Journal & conference	Abstract ("code clone")and ("Smart contract") and ("Blockchain")	10 Years
2	Science Direct	www.sciencedirect.com	Journal & conference	Abstract ("code clone")and ("Smart contract") and ("Blockchain") and ("Ethereum")	All date
3	Springer	www.springer.com	Journal & conference	Abstract ("code clone")and ("Smart contract") and ("Blockchain") and ("Ethereum")	All year's entire range of publication
4	Google Scholar	www.googlescholar.com	Journal & conference	("code clone")and ("Smart contract") and ("Blockchain" ,"Ethereum")	2007-2022

3.2 Selection results

The following outcomes, divided into by platform, were found using the keywords:
Google Scholar : Total results 22 paper
IEEE : Total results 20 paper
Springer : Total results 5 paper
Science Direct : Total results 7 paper
After this primary study 23 papers were left over for study.

IV. 4. RESULT

The following research questions are addressed in the paper :
RQ1 :What is the current status of Smart contract clone detection?
RQ1.1:What methods are used for smart contract clone detection ?
RQ2.What tools are available for smart contract clone detection?
RQ3.Which Machine Learning and deep learning techniques used for smart contract clone detection ?

RQ1 Current status of Smart contract clone detection

Ethereum has grown in popularity as a trustworthy platform for conducting commerce and managing finances on the Blockchain. However, Ethereum smart contracts contain a significant security issue. Smart contracts have been plagued by claims of faults and weaknesses, which not only complicate blockchain maintenance but can result in significant losses. Better tools are required for software developers if they are to successfully validate and verify the accuracy of smart contracts. In this article, we argue in favor of utilizing SMARTEMBED[8], a web service tool that can assist Solidity developers in identifying potential problems with cloned smart contracts as well as instances of duplicate contract code. The core of our technique is comprised of different approaches for comparing codes and code embeddings. Our capacity to identify code clones and issues that are associated with clones can help users feel more confident in the code that they have contributed to our project. In order to establish whether or not there is any overlap, the vectors that were utilized in the process of embedding the code in the Ethereum blockchain were compared to those that were connected with known vulnerabilities. There have been a lot of research done in the past that have looked into the issue of detecting faults in smart contracts. Current approaches have a number of drawbacks, one of the most significant of which is that they rely excessively on human-provided bug patterns and specification requirements. It may be prohibitively expensive and time-consuming to continually develop new rules and design new tests to address newly discovered defects and vulnerabilities generated by attackers. This is because the immense stakes involved in smart contracts, as well as the continuing fight between attackers and defenders, make it necessary for smart contracts to be constantly updated. Recent papers [4] have focused on the identification of clones in Ethereum smart contracts as well as the existence of clones themselves.

Because the symbolic transaction sketch or pair-wise comparisons are so expensive, they can only be utilized for clone detection. Machine learning and deep learning approaches have been utilized to address clone identification and bug detection concerns in traditional software programs

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 7, July 2023

[6] but there hasn't been much success in this area with smart contracts.

Clones can be found in a variety of ways. The majority of techniques are strongly reliant on their source representation and match detection mechanism.

RQ1.2 Intermediate source representations and match detection techniques

During pre-**processing**, first, the source code is cleaned up by removing comments and whitespace. An intermediate representation of the preprocessed code was built using the appropriate transformation techniques. Clones have varying degrees of granularity across different intermediary source representations. All of these items are detailed in Table 1. The two most prevalent types of intermediate transformations are XML parse trees and abstract syntax trees (ASTs). Code, control flow, and the structure of an XML document are all represented by the AST, CFG, and XML Parse Trees, respectively. Syntactic clones, or chunks of code that share the same syntax, are found using ASTs [7,15,19]. This includes duplicated statements, contracts, and complete functions. Clones can be recognized at various levels of detail, such as in contracts and assertions, using CFG[3,6]. However, CFG-based clone identification is based on comparing their control flow pattern. XML parse trees [4, 8, 10] are used to represent the structure of XML documents. An XML parse tree is a tree-like structure that represents the hierarchical structure of XML elements, their attributes, and their content. This representation can be used to detect clones in XML documents, which are fragments that have similar structures, including similar element names, attributes, and content.

TABLE 1 : Intermediate representation

Intermediate representation/transformation technique	Clone granularity level	References
control flow graph (CFG)	Smart contract	[3,6]
Abstract syntax trees (ASTs)	Smart contract, Function ,statements	[7,15, 19]
XML Parse Tree	Smart Contract	[4,8,10]

Match detection methods are a major issue in the clone detection process. Table 2 includes every method for detecting matches that we discovered. Primary research that compares and contrasts the various matchmaking algorithms is provided. The most prevalent match detection methods are Symbolic Transaction Sketch, Code Parsing, XPath Matching, Substring Comparison, and Longest Common Subsequences (LCS). In [3,6] Symbolic transaction sketch is used for code clone type 4 detect in smart contract. Code Parsing involves parsing code to generate an abstract syntax tree, which can then be used to identify similarities between different code fragments. XPath Matching involves using XPath expressions to extract and compare specific code elements, such as function calls or variable assignments. Substring Comparison involves comparing the substrings of code fragments, while LCS[23] involves finding the longest common subsequence between code fragments.

TABLE 2 : Match Detection

Match Detection Technique	Clone granularity level	References
Symbolic Transaction Sketch	Smart contract	[3,6]
Code parsing	Contract, Function, Statement	[7,8,11]
XPath Matching	Smart contract	[4]
Substring Comparison	Smart contract	[10]
Longest common subsequences (LCS)	Smart contract	[23]

RQ2.smart contract clone detection Tools

Several techniques and tools are used to detect cloned smart contracts.

It turns out that Eclone [3] was used. EClone is an Ethereum-based semantic clone detection that makes use of Symbolic Transaction Sketch. This schematic depicts a variety of salient semantic features resulting from symbolic interactions.In order to facilitate the subsequent computation of similarity, they normalized two smart contract designs into numeric vectors of the same length.

Smartcheck [4] analyses the smart contract using a lexical and syntactical technique. We use a proprietary Solidity language and ANTLR (a parser generator) to generate an XML parse tree to get to this intermediate form. XPath searches are used to parse the intermediate form and detect vulnerability patterns.

SmartEmbed[7], a machine learning and deep learning tool based on code embeddings and similarity testing methodologies, was proposed by Gao et al. in 2019. By comparing the code embedding vectors of the current Ethereum code to those of known defects, this tool may detect code clones and clone-related problems.

Coinwatch[10] proposed a systematic remedy to solve the cloned cryptocurrencies, called COINWATCH (*CW*) . System based technology that uses code evolution analysis and clone detection to identify potentially vulnerable coins.

Eth2vec[15], a method for static analysis that is based on machine learning, has been used by the researchers. Through the study of assembly language, EVM bytecode, and abstract syntax trees, this compiler has the ability to generate code for smart contracts. The vulnerabilities in smart contracts have been uncovered with the use of this technology. Eth2Vec is a rewrite-resistant, high-throughput code generator. The targeted smart contract code is sent into the natural language processing analysis tool Eth2Vec, which subsequently reports on the presence and type of vulnerabilities in the code. Users of Eth2Vec are able to study the source code of smart contracts in a quick and straightforward manner, without the need for prior understanding of smart contract vulnerabilities.

For example, the most recent official exchange rate for the US dollar is data that some smart contracts require but is not kept in the blockchain. As a means of achieving this goal Oracles are complex smart contracts designed for the Ethereum platform. Oracle smart contracts grant external services access to the Oracle as well as the power to change its status. As a result, Oracles can serve as reliable intermediates between other smart contracts and the real world. Rather of relying on an external service, a non-Oracle smart contract will communicate directly with the Oracle smart contract. An Oracle transaction will be sent in by an external service in the case that the event is completed successfully.

Because of this, we went looking for a device that could (i) detect clones at a price that was reasonable, and (ii) be customized so that it met our particular needs.

NiCad [21]: NiCad is a piece of software that scours the smart contracts on Ethereum for instances of duplicated or duplicated-looking code. The goal of this research is to gain an understanding of the ways in which a variety of smart contracts in relevant industries make beneficial use of code cloning and near-miss detection. NiCad is ideally suited for use in the construction of a model-driven development framework for decentralized applications (DApps) due to its capacity to identify code clones of Type 1, Type 2, Type 2c, Type 3-1, and Type 3-2c. [21] Out of all the free clone detection apps that we tested, NiCad stood out as the most flexible and user-friendly option. NiCad is a well-known program for locating clones that are virtually indistinguishable from one another and is based on the textual analysis of digital data. Clone detection studies have made considerable use of it as a result of its excellent precision and recall for recognizing practically identical clones. This is because of its ability to distinguish between nearly identical clones [21].

Table 3 : Smart contract clone detection tools

Tool Name	**Clone Type**	**Technique**	**Parameters Performances**	**Experimental Datasets**	**Evaluation**	**Ref.**
Eclone	Type IV	Symbolic execution	Accuracy , ROC curve , Baseline Precision ,threshold, TP , FP	2,117 Solidity smart contracts	Identifying Clones with accuracy 93.27%	[3]
SmartEmbed	Type III,IV	Code parsing	Precision ,Recall, Threshold, F1 Score , False Positive rate ,False Negative rate	More than 22,000 smart contracts	The tool identified 90% of the Clone ratio.	[7]
CoinWatch	Type I ,II,III	Substring compression	Accuracy , TP vs FP	1094 cryptocurrencies	786 identify true Vulnerability in 384 Projects	[10]
SmartCheck	Type III,IV	Static code analysis	TP , FP , FN ,FDR(False discovery rate) , FNR (False negative rate).	4,600 verified smart contracts.	The tool detected 99.9% of clone contracts.	[4]
Eth2Vec	Type III,IV	Static code analysis	Precision ,Recall ,F1-score	5,000 contract files from Etherscan	This technique may detect vulnerabilities in rewritten code with an accuracy of 77.0% and identify reentrancy with 86.6% precision by incorporating lexical semantics between contracts and extracting features implicitly.	[15]

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 10, Issue 7, July 2023

RQ3.Which Machine Learning and deep learning techniques are used for smart contract clone detection?

Machine learning has developed an effective technique for detecting smart contract clones. Machine learning algorithms can be trained to identify similarities and differences between code fragments as well as potential code clones by using supervised, unsupervised learning approaches and deep learning.

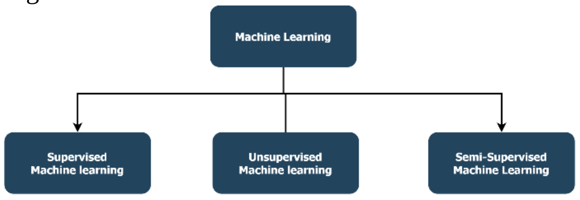

Machine Learning
Supervised Machine learning
Unsupervised Machine learning
Semi-Supervised Machine Learning

Figure 4: Machine learning classification methods

Supervised learning is the process of training a machine learning model on labeled datasets, with each data point classed as either clone or non-clone. The model can then be used to classify new code fragments based on their similarity to the training data. Common supervised learning methods for smart contract clone detection include Support Vector Machines (SVMs) [15] and neural networks. Unsupervised learning involves training a model on unlabeled datasets, in which the model must identify patterns and relationships on its own. Clustering approaches such as K-means and hierarchical clustering can group similar code snippets without first determining whether ones are clone.

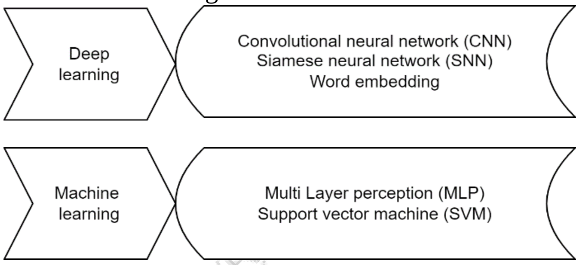

Deep learning
Convolutional neural network (CNN)
Siamese neural network (SNN)
Word embedding
Machine learning
Multi Layer perception (MLP)
Support vector machine (SVM)

Figure 5: Recent research uses a category-based grouping of classification algorithms.

This category includes both supervised and unsupervised approaches. Code clone detection is just one of the many machine learning and deep learning technologies used in smart contracts. Researchers use a wide range of approaches, including Siamese neural networks, convolutional neural networks, multi-layer perceptrons, support vector machines (SVM), and word embeddings. In [18,19,21] paper researchers use deep learning techniques (CNN, word embeddings, and SNN). CNNs have been used for bug detection, clone detection, and code clustering, while MLPs have been applied to code clone detection and code search. Word embedding techniques, which involve mapping words to high-dimensional vectors, have been used for the unsupervised detection of clones and bugs in smart contracts.

TABLE 4: Machine learning and Deep learning techniqueMachine Learning technique

	Type of machine learning	**Category**	**Application**	**Performance parameters**	**References**
Convolutional neural network (CNN)	Supervised	Deep Learning	Bug detection, clone detection, and code clustering	Precision, Recall F1-score, ARI	[19]
Multi-Layer Perceptron (MLP)	Supervised	Machine Learning	Code Clone Detection and Code Search	Accuracy	[18]
Word Embedding Technique	Unsupervised	Deep Learning	Detect the clone and bug in Smart Contract	Precision, Recall, Threshold, F1 Score, False Positive rate, False Negative rate Clone ratio 90%	[8]
Support vector machine	Unsupervised	Machine Learning	Detects vulnerabilities in Ethereum smart contracts.	Precision Recall F1-score	[15]
Siamese neural network	Supervised	Deep Learning	Detect the similarity of Ethereum smart contracts	Precision, Recall, F1-score , ROC, TP, FP F1-score :0.9850 Accuracy: 98.37%	[21]

The most frequent method for confirming smart contracts utilizing structural code embeddings is to use deep learning and machine learning. Because of its broad applicability, our approach can be applied to a wide range of code debugging and maintenance jobs, including those that necessitate the use of code embedding or similarity testing. Among these are the detection of duplicate (or "cloned") [8,9,10,21] contracts, the detection of specific sorts of faults in a large contract corpus, and the comparison of a contract to a database of known problems. Furthermore, by constructing code embeddings for the increasing bug patterns, our method may simply incorporate new bug-checking criteria, removing the need for any further manual work in building the bug specifications. SmartEmbed [7] to assist Solidity writers in examining their own smart contracts for code duplication and flaws. This was done after taking into account the Solidity community's criticism of Github. Furthermore, we enhance SmartEmbed in three ways to meet the efficiency needs of a web-based tool: (i) use a matrix computation to substitute numerous discrete loop structure calculations. (ii) Cache the code embeddings to avoid loading the same information multiple times. (iii) To speed up data retrieval, we will develop smart contract indexes in our database.

V. CONCLUSION

The host and execution environment for smart contracts is Ethereum, a blockchain platform. Digital currencies and initial coin offerings (ICOs) would not exist if smart contracts were absent. It is essential to address the issue of security in Ethereum smart contracts. In contrast to more conventional approaches to software development, smart contracts cannot be modified after they have been created. Because of their limitations and flaws, smart contracts leave themselves open to the possibility of suffering catastrophic financial loss. Creators of smart contracts can avoid writing incorrect code by borrowing code and functionality from established libraries and frameworks like OpenZeppelin.Our research contributes to efforts to increase the reuse capabilities of Solidity and, more broadly, smart contract programming languages. When creating such reuse mechanisms, tool developers and language engineers may find this work useful.

Blockchains present a new processing paradigm for distributed systems where data must be stored indefinitely. Smart contracts are computer programs that can be run on a blockchain. Once installed, these programs cannot be changed and will run eternally for the duration of the platform's existence. Because the deployed code is immutable, any modifications must be done in the source code and then re-deployed. As a result of insufficient software engineering practices, blockchains represent larger risks than traditional software environments. Given that the great

majority of presently implemented smart contracts are designed for monetary purposes, any security issues they may include could have serious financial ramifications.

This paper can be used by business stakeholders to better analyze the technical potential and security risks of blockchain systems. Unfortunately, due to a lack of communication among engineers, such initiatives are particularly difficult. When it comes to blockchain DevOps operations, we anticipate an increase in the use of automated audit procedures during the phase of integration (pre-deployment) [1]. Such a growth would be beneficial for a number of reasons. One example of a solution that might be used to propose refactoring suggestions for reducing the clone ratio in distributed code and, as a result, improving the quality of the platform's code is the NiCad-based tool that was exhibited in this research. This tool is only one example of a solution that may be utilized. We also anticipate that platform agents will start offering quality control as a service for a cost that is proportional to the work necessary to complete the computation.

Because of the difficulties involved in generating smart contracts, future research is required into how typical software engineering lifecycle models might be modified. It is essential for future research to broaden the scope of this study to incorporate the smart contract programming languages of other platforms, such as Bitcoin's Script language. This is one of the most important areas to investigate.

REFERENCES

[1] Roy, C.K. and Cordy, J.R., 2007. A survey on software clone detection research. *Queen's School of Computing TR*, *541*(115), pp.64-68.

[2] Rattan, D., Bhatia, R. and Singh, M., 2013. Software clone detection: A systematic review. *Information and Software Technology*, *55*(7), pp.1165-1199.

[3] Liu, H., Yang, Z., Liu, C., Jiang, Y., Zhao, W. and Sun, J., 2018, October. Eclone: Detect semantic clones in ethereum via symbolic transaction sketch. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 900-903).

[4] Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E. and Alexandrov, Y., 2018, May. Smart check: Static analysis of ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain* (pp. 9-16).

[5] Kiffer, L., Levin, D. and Mislove, A., 2018, October. Analyzing Ethereum's contract topology. In *Proceedings of the Internet Measurement Conference 2018* (pp. 494-499).

[6] Liu, H., Yang, Z., Jiang, Y., Zhao, W. and Sun, J., 2019, May. Enabling clone detection for ethereum via smart contract birthmarks. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)* (pp. 105-115). IEEE.

[7] Gao, Z., Jayasundara, V., Jiang, L., Xia, X., Lo, D. and Grundy, J., 2019, September. Smartembed: A tool for clone and bug detection in smart contracts through structural code embedding. In *2019 IEEE International Conference on*

Software Maintenance and Evolution (ICSME) (pp. 394-397). IEEE.

[8] Gao, Z., Jiang, L., Xia, X., Lo, D. and Grundy, J., 2020. Checking smart contracts with structural code embedding. *IEEE Transactions on Software Engineering*.

[9] Kondo, M., Oliva, G.A., Jiang, Z.M.J., Hassan, A.E. and Mizuno, O., 2020. Code cloning in smart contracts: a case on verified contracts from the Ethereum blockchain platform. *Empirical Software Engineering*, *25*(6), pp.4617-4675.

[10] Hum, Q., Tan, W.J., Tey, S.Y., Lenus, L., Homoliak, I., Lin, Y. and Sun, J., 2020, November. CoinWatch: A clone-based approach for detecting vulnerabilities in cryptocurrencies. In *2020 IEEE International Conference on Blockchain (Blockchain)* (pp. 17-25). IEEE.

[11] Gao, Z., 2020, December. When deep learning meets smart contracts. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (pp. 1400-1402).

[12] He, N., Wu, L., Wang, H., Guo, Y. and Jiang, X., 2020, February. Characterizing code clones in the Ethereum smart contract ecosystem. In *International Conference on Financial Cryptography and Data Security* (pp. 654-675). Springer, Cham.

[13] Di Angelo, M. and Salzer, G., 2020, September. Characteristics of wallet contracts on Ethereum. In *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)* (pp. 232-239). IEEE.

[14] Chen, J., Xia, X., Lo, D., Grundy, J. and Yang, X., 2021. Maintenance-related concerns for post-deployed Ethereum smart contract development: issues, techniques, and future challenges. *Empirical Software Engineering*, *26*(6), pp.1-44.

[15] Ashizawa, N., Yanai, N., Cruz, J.P. and Okamura, S., 2021, May. 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* (pp. 47-59).

[16] Hu, T., Liu, X., Chen, T., Zhang, X., Huang, X., Niu, W., Lu, J., Zhou, K. and Liu, Y., 2021. Transaction-based classification and detection approach for Ethereum smart contract. *Information Processing & Management*, *58*(2), p.102462.

[17] Pierro, G.A. and Tonelli, R., 2021, March. Analysis of source code duplication in Ethereum smart contracts. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 701-707). IEEE.

[18] Cui, N., Jiang, Y., Gu, X. and Shen, B., 2022. Zero-Shot Program Representation Learning. *arXiv preprint arXiv:2204.08360*.

[19] Yang, S., Gu, X. and Shen, B., 2022. Self-Supervised Learning of Smart Contract Representations.

[20] Cui, N., Jiang, Y., Gu, X. and Shen, B., 2022. Zero-Shot Program Representation Learning. *arXiv preprint arXiv:2204.08360*.

[21] Tian, Z., Huang, Y., Tian, J., Wang, Z., Chen, Y. and Chen, L., Ethereum Smart Contract Representation Learning for Robust Bytecode-Level Similarity Detection.

[22] Khan, F., David, I., Varro, D. and McIntosh, S., 2022. Code Cloning in Smart Contracts on the Ethereum Platform: An Extended Replication Study. *IEEE Transactions on Software Engineering*.

[23] Samreen, N.F. and Alalfi, M.H., 2022. Mining Domain Models in Ethereum DApps using Code Cloning. *arXiv preprint arXiv:2203.007*

IFERP connecting engineers...developing research