Informatics Institute of Technology

In Collaboration With

University of Westminster, UK

University of Westminster, Coat of Arms

Automated Smart Contract Security Auditing System

A Project Specification & Product Design

Mr. Danuja Silva

W1761772 / 2018221

Supervised by

Mrs Minoli de Silva

This Project Proposal is submitted in partial fulfilment of the requirements for the BSc (Hons) Computer Science degree at the University of Westminster

16th February 2023

ABSTRACT

Over the past few years, smart contract development has been growing exponentially, leading to powerful decentralized applications utilized in many crucial high-value fields such as finance and asset transfer. Despite all these advantages, smart contracts can be vulnerable to exploits, which may result in substantial financial losses and hurt the overall blockchain ecosystem. As a result, the need for better security auditing systems for smart contracts has also risen. Existing security auditing tools lack scalability since they rely on predefined rules from experts or check for similar keywords to detect vulnerabilities making them less effective and causing high false detection rates.

ASCSAS helps bridge these gaps by utilizing deep ensemble learning techniques, improving the overall detection of vulnerabilities present in smart contracts. ASCSAS follows different feature engineering and feature selection techniques that have not been considered in current literature and the experimental studies conducted on different datasets prove that the proposed system approach will be able to perform multi-class classification for the various types of vulnerabilities present on smart contracts, demonstrating its overall effectiveness.

Keywords : Blockchain, smart contracts, deep learning, ensemble learning

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Acronym	Description
DL	Deep Learning
ML	Machine Learning
AI	Artificial Intelligence
NN	Neural Networks
DNN	Deep Neural Network
DR-GCN	Degree Free Graph Convolutional Network
TMP	Temporal Message Propagation
LSTM	Long short-term memory
TCN	Temporal Convolutional Network
CNN	Convolutional Neural Network

RNN	Recurrent Neural Network
SOTA	State of the art
GAN	Generative Adversarial Network

Table 1 : List of Abbreviations

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Chapter Overview

The main motive of this chapter is to provide the reader with an introduction to smart contracts used in the blockchain, vulnerabilities in smart contracts and prevailing vulnerability detection tools. This will be used to recognize current issues faced with the state of the art and justify why effective and scalable detection methods are crucial to detect vulnerabilities in smart contracts.

1.2 Problem Domain

1.2.1 Smart Contracts

Smart contracts are pieces of code that are automatically developed and executed on the Ethereum platform. They are executed when specific business conditions are met to regulate the movement, storage, and exchange of data that can be found in standard agreements. These contracts will be written using Solidity, one of the most popular and high-level programming languages and will be compiled into its bytecode format that will be permanently stored at a unique blockchain address (Tikhomirov et al., 2018). Smart contracts can automate simple contract structures such as transactions procedures, eliminating the need for third-party oversight, which is crucial in the blockchain environment (Introduction to smart contracts, 2022). And they can build more complex structures such as forming organizations without any central authority. The recent breakthrough in blockchain and smart contract creation has resulted in many studies been done on smart contract security and maintenance.

1.2.2 Smart Contract Vulnerability Detection Systems

As more smart contracts are being developed, the demand for smart contract vulnerability detection systems grows. Traditional vulnerability detection systems often use rule-based or machine learning-based methodologies to discover specific vulnerabilities based on expert-defined criteria or attributes and characterize these vulnerabilities(Qian et al., 2022). This may be a time-consuming and tiresome effort and the possibility of missing out other key features increases when

detecting certain vulnerabilities, but by employing deep learning technologies, vulnerability patterns can be learned and detected without the need for expert-defined rules. Deep learning has brought several advantages in automating detection techniques especially when it comes to vulnerability detection. (Li et al., 2019).

1.3 Problem Definition

With millions of smart contracts being deployed on the network, security flaws have emerged as a key danger in smart contracts, and vulnerability detection systems require further development (Gopali et al., 2022). Traditional vulnerability detection methods use predefined rules set by experts to detect vulnerabilities. This is proven to be unreliable and limited in scalability since they only rely on a simple similarity checking procedure which searches for similar terms and keywords and new unseen vulnerabilities will not be detected (Ye et al., 2022).

Moreover, due to constant change in structures or semantics of languages or frameworks used in smart contract development, building a detection system that can be comprehensive enough to detect newly wedged vulnerabilities by combining different dynamic analysis techniques and static analysis is a challenge (Qian et al., 2022).

1.3.1 Problem Statement

Existing vulnerability detection systems are limited in scalability because they are built based on expert defined rules to detect vulnerabilities making it difficult to detect new vulnerabilities with high accuracy in smart contracts.

1.4 Research Questions

RQ1: What are the recent advancements in ML/DL that can be considered when building a hybrid vulnerability detection system for smart contracts?

RQ2: How can a combination of static and dynamic analysis be used for effectively detecting vulnerabilities?

RQ3: How can a hybrid detection system be able to detect vulnerabilities accurately?

1.5 Research Aims and Objectives

1.5.1 Research Aim

The aim of this research is to design, develop & test a unified, scalable vulnerability detection system that will aid in the development of secure smart contracts by accurately detecting existing and potential vulnerabilities.

To further elaborate on the aim, this research project will create a system that combines static and dynamic analytic approaches utilizing both source code and bytecode of a smart contract with ensemble models to accurately detect vulnerabilities. This is possible by combining existing state of the art technologies with various improvements to the models and developing a scalable architecture.

1.5.2 Research Objectives

The research objectives of the project will be summarized below.

Research Objectives	Explanation	Research Questions	Learning Outcome
Literature Review	Reading and critically evaluating previous work done on the related research areas. RO1: To analyze the blockchain domain and how smart contracts are used in blockchain and the security flaws with smart contracts. RO2: To review current software vulnerability detection systems and vulnerability detection systems used for smart contracts and critically evaluate them based on their advantages and disadvantages. RO3: To identify research gaps in general software vulnerability tools and finding which ones can be	RQ1, RQ2, RQ3	LO4, LO2, LO5

	applicable in the context of detecting vulnerabilities in smart contracts.		
Data Gathering and Analysis	Summarizing the requirements for the project using the appropriate tools and frameworks to fulfill the mentioned research gaps & challenges. RO1: Gather information of the blockchain, smart contract, smart contract vulnerabilities and general software vulnerabilities found in different languages. RO2: Gather the requirements to build an effective vulnerability detection model based on gaps found from current literature. RO3: Gather insights from domain and technological experts on smart contract vulnerabilities and other important factors to build an effective vulnerability detection system.	RQ1, RQ2	LO2, LO7, LO5, LO1
Design	Designing a system that can solve the above-mentioned problems. RO1: To design a code segment classifier using ensemble classifiers for the DL model. RO2: To design the hybrid model that leverages the ensemble classifiers with the model. RO3: To design a method that can effectively concatenate the models without hindering performance.	RQ1, RQ2	LO1
Implementation	Developing the vulnerability detection system according to the designed architecture.	RQ2, RQ3	LO1, LO5

	RO1: To develop initial classification models which will be used for developing and evaluating the vulnerability detection system. RO2: To develop the fundamental functionalities by using relevant hardware and software requirements. RO3: To integrate all the models in a scalable architecture that can be used for adding in and training new models and present it in a web-based application.		
Testing, and Evaluation	To test and evaluate the prototype by domain experts/ researchers. RO1: Creating test plans for functional testing and performance testing. RO2: Evaluate the novel model by benchmarking with other existing models.	RQ3	LO4

Table 1: Research objectives

1.6 Research Novelty

1.6.1 Problem Novelty

Due to the increased use of blockchain technology and smart contracts, detecting vulnerabilities in smart contracts has become very important. However, traditional detection methods are unreliable, less accurate and limited in scalability, as they detect vulnerabilities based on predefined expert rules and do not consider the semantic properties of smart contracts. Currently, there is a need for a new approach to smart contract vulnerability detection that is accurate, scalable, and efficient.

1.6.2 Solution Novelty

The proposed research project aims to build a unified and scalable smart contract vulnerability detection system which has not been attempted before by utilizing different ensemble learning techniques. This unique approach can aid in the development of secure smart contracts and improve the overall blockchain ecosystem.

1.7 Research Gap

After reviewing the current literature, several limitations were identified. They are summarized as follows:

- Static analysis is performed at the bytecode or source code level, it misses dynamic interaction with external contracts or might not take alternative execution pathways into account, which might result in the identification of false vulnerabilities.
- Dynamic analysis techniques, such as fuzzing methods, necessitate the creation of several well-generated test cases that allow for the monitoring of any unexpected behavior throughout execution time, which is a complicated procedure and might not be able to cover all possible test paths. They may also be unable to capture any specific code fragments from where the discovered vulnerability is occurring (Qian et al., 2022).
- Most vulnerability detection systems do a simple similarity check for similar keywords which are defined by experts and determine if they are a vulnerability or not. This can flag potential safe contracts as unsafe, leading to unsatisfactory detection accuracy and high false alarm rates (Xu, Liu, and Zhou, 2022).

As discussed above, the most recent research on vulnerability detection systems demonstrates that there are both empirical and performance gaps.

Therefore, the goal of this research is to present and test a novel vulnerability detection approach that can fill the mentioned gaps, speeding up the engineering process and assisting in the creation and implementation of secure smart contracts.

1.8 Contribution to The Body of Knowledge

The contributions of this research can be summarized as follows:

- Vulnerability Detection systems: Data science[(ML)+(DL)] + Ensemble models

- Developing safe smart contracts: Vulnerability detection + AI + Automation

1.8.1 Technological contribution

Developing a unified, scalable hybrid vulnerability detection approach that will mix static and dynamic analysis to increase performance and accuracy when finding vulnerabilities in a way that has not been explored in earlier studies. This will automate numerous detection techniques, reducing the time required for developers/experts in developing secure smart contracts. This novel detection architecture can be aided for the development of other types of detection systems and detecting vulnerabilities for other languages.

1.8.2 Domain contribution

Smart contracts are the next building blocks in the blockchain, having the ability to grow decentralized systems to the next level by automating many predefined procedures or allowing contracts to be executed in complete transparency with no outside interference. All smart contracts must be properly audited for security vulnerabilities, and a detection system will aid in the process of finding and correcting potential vulnerabilities or exploits created by developers/experts reducing time for manually reviewing code. This, in turn, will aid in the development of secure smart contracts and the widespread acceptance of the technology.

1.9 Research Challenge

The primary purpose of this research is to solve the present limitations of SOTA vulnerability detection techniques. This is accomplished by developing an innovative and scalable vulnerability detection architecture to increase overall accuracy and discover any new potential attacks in smart contracts. There will be numerous research challenges based on the research strategy in different areas, which are listed down.

- The Ethereum ecosystem is relatively new, so it lacks proper procedures for collecting and structuring vulnerabilities (Rameder, di Angelo and Salzer, 2022)

- Currently, smart contracts can be written using different programming languages such as Solidity, Go, Java, and so on. This results in different code structures and different

semantics for contracts and building a detection system to adapt to these languages will be a challenging task (Qian et al., 2022)

- The lack of consistent and standardized datasets reduces the overall effectiveness of DL-based smart contract vulnerability detection systems and restricts their capacity to only detecting specific vulnerabilities (Zhang et al., 2022).

1.10 Chapter Summary

This chapter presented the problem being solved by the research that is to be conducted for upcoming months. A brief introduction to smart contracts and problems faced and current issues in vulnerability detection systems in smart contracts have been discussed. Moreover, a summary of existing work with a critical evaluation, the research challenges the author will face have also been discussed in detail with relevant methodologies and strategies while fulfilling the research aims and objectives.

CHAPTER 2: SOFTWARE REQUIREMENT SPECIFICATIONS

2.1 Chapter Overview

This chapter will focus on gathering out all the system requirements, identifying all the possible stakeholders, presenting all the requirement gathering techniques used and prioritize the requirements to make an optional solution for the problem. First, a rich picture diagram is presented, alongside all the responsible stakeholders and their responsibilities in the system. All the requirement gathering techniques used are discussed with factors such as why each technique was chosen and summary of all the findings. Finally, in the latter section, a use case diagram and descriptions, as well as all of the systems functional and non-functional requirements have been specified.

2.2Rich Picture Diagram

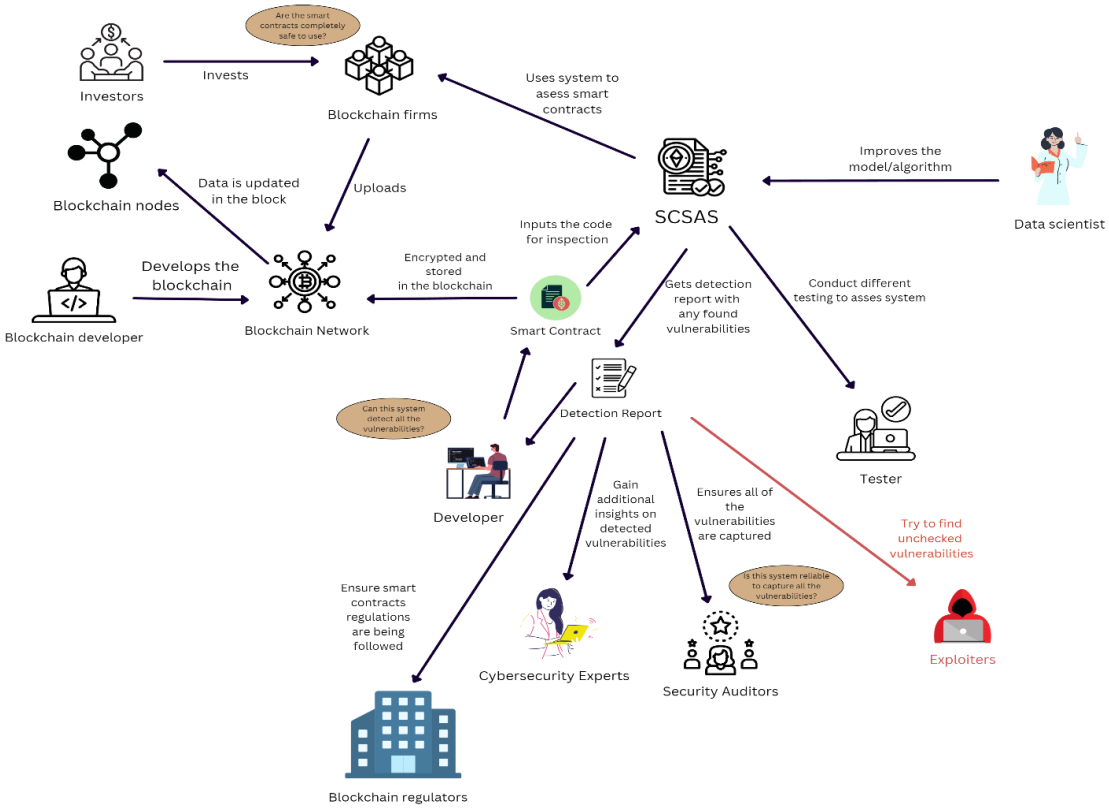

Figure 1 : Rich picture diagram(self-composed)

2.3 Stakeholder analysis

The stakeholder onion model will depict all stakeholders who are associated with the system, along with a brief explanation of each stakeholder's connection to the system, in stakeholder viewpoints.

2.3.1 Stakeholder onion model

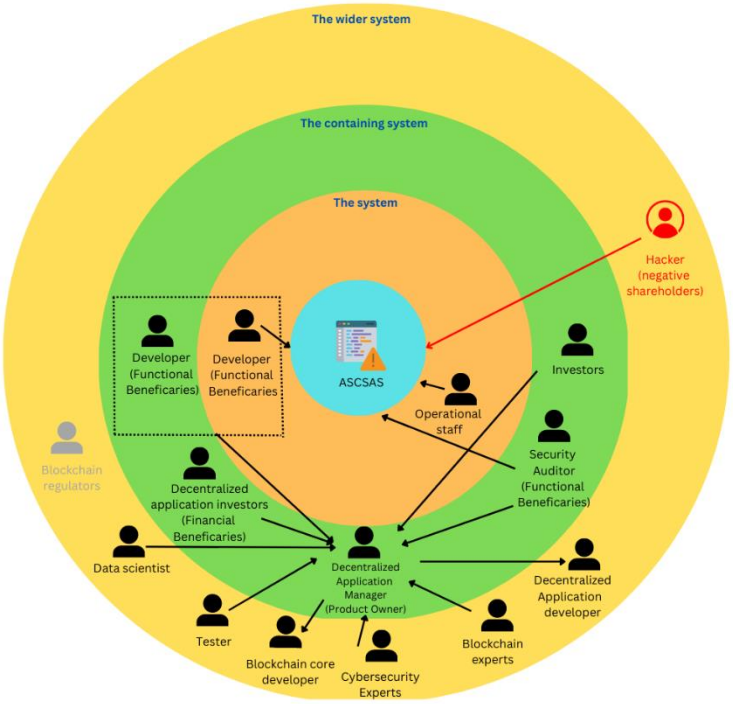

Figure 2 : Onion model(self-composed)

2.3.2 Stakeholder viewpoints

Table 2 : List of stakeholders & respective viewpoints

Stakeholder	Role	Description

Developer	Functional Beneficiaries	They are responsible for writing smart contracts that will be deployed on the blockchain platform. They can use ASCSAS to detect any unseen vulnerabilities before deploying it to the blockchain.
Security auditors		Asses any security threats and ensures cybersecurity best practices and policies are followed when developing smart contracts. A system like ASCSAS can help security auditors do a thorough analysis of a smart contract to ensure that it is safe.
Decentralized applications manager/owner	System/product owner	People that are responsible for the maintenance, administration, and development of Dapps. They can ensure that the applications built are secured and comply with legal regulations.
Investor	Financial beneficiaries	People or companies that invests on the development of blockchain technology. Investors can ensure that the Dapps they invest are safe from any vulnerabilities, minimizing the risk of any losses.
Decentralized applications investors		People or organizations that invests especially on blockchain related firms.
Blockchain experts	Expert, Quality regulators	People responsible for providing expertise & domain knowledge of the blockchain, which can be utilized to improve the system.
Cybersecurity experts		Provide expertise in terms of security related concepts or protocols that need to be followed. These experts can use a system like ASCSAS to enhance their knowledge on smart contract security.
Blockchain regulators		.Organizations that ensure proper regulations are followed with the use of blockchain technology and cryptocurrencies. They can ensure Dapps built using smart contracts follow proper regulatory requirements related to security.

Tester	Quality inspector	Conducts different tests to ensure system is fully operational without any errors.
Operational staff	Maintenance staff	Ensure the system is running without any errors.
Blockchain core developer	Maintenance, support team	People who design, maintain, and build the decentralized application.

2.4 Selection of Requirement Elicitation Methodologies

Multiple requirement elicitation methodologies such as literature review, interviews and prototyping have been followed to complete the development of this research project. All the reasons for selecting these techniques have been justified below.

Method 1 : Literature Review
A detailed review of existing tools and technologies used in the research domain was undertaken, allowing the author to understand the current state of the art, alongside all the crucial key concepts and terminologies used in the research project and identify potential areas where further research was needed.
Method 2 : Interviews
Interviews were identified as a good requirement gathering technique due to the research domain being niche and new. One-on-one interviews with various stakeholders such as blockchain developers, blockchain experts and security auditors were necessary to confirm the current limitations with the state of the art. This also allowed the author to obtain qualitative feedback and gain additional insights on some of the requirements for the proposed solution.
Method 3 : Prototyping
Prototyping was used to validate the solution, identify potential challenges, experiment with different algorithms and design options and create a low-fidelity model(core component of the prototype) and get feedback from users. This allowed the author to get a better understanding of the problem and identify any areas that needed improvement.

Table 3 : List of the requirement elicitation methodologies been used.

Method 1 : Survey

Due to the research domain being relatively new and limited people being interested in the field of smart contract development and security, surveys would not be feasible enough to gather accurate and reliable information.
Method 2 : Brainstorming
To conduct an effective brainstorming session, a set of diverse professionals are needed. However, since the domain of smart contract security requires expertise in both blockchain technology and smart contract security, it would be challenging to gather a suitable team capable of conducting a successful brainstorming session with the given time constraint.

Table 4 : Justification as to why other requirement elicitation methodologies were not used.

2.5 Presentation of the Outcome through Elicitation Methodologies

2.5.1 Literature Review(LR)

A summary of the findings from the literature is presented down below with relevant citations.

Citation	Findings
(Qian et al., 2022)	It was identified that existing vulnerability detection systems only focus on detecting few vulnerabilities, reducing the systems overall coverage. A system that can detect various types of vulnerabilities will be crucial as it will aid developers and the overall blockchain ecosystem
(Xu, Liu, and Zhou, 2022)	Most of the systems only focus on checking for similar code fragments or bytecodes without considering the semantic meaning of the code, leading to high false detection rates.
(Shakya et al., 2022)	While most systems focus on detecting vulnerabilities only using the bytecodes of contract. Using a combination of both bytecode and source code of a smart contract for analysis can yield to better detection results.
(Matloob et al., 2021)	Utilizing ensemble learning techniques can reduce the overall false negatives and false positives rather than using individual classifiers, leading to better accuracy

Table 5 : Literature review findings

Current findings suggest that most of the existing systems are not capable of detecting newer vulnerabilities. This is because they heavily on human experts to define rules or features to detect vulnerabilities. This often leads to potential issues such as lack of scalability and poor detection results, often causing high false detection rates. As a result, making it difficult for developers to differentiate between real and falsely detected vulnerabilities. To overcome these current limitations, the use of machine learning techniques has been explored to reduce the overall need for expert defined rules to identify vulnerabilities by automating the vulnerability detection process. This will improve detection results and potentially increase system scalability.

2.5.2 Interviews

Semi-structed interviews were conducted with a panel of experts to obtain qualitative feedback both on technical and domain expertise for the proposed solution. The panel consisted of 2 Blockchain developers and 1 security auditor to get domain expertise. In addition, 2 data scientists and 1 research assistant to get technical expertise. Following table presents a thematic analysis of the interviews that were conducted.

Code	Theme	Conclusion
• Dataset • Smart contract data • Less data for training • Labeling vulnerabilities	Dataset availability	Since the availability of proper data plays a critical role in any data science project, most of the interviewers were concerned about the availability of smart contract data and vulnerability labeling to capture the vulnerabilities. Some suggested to use the official Ethereum website and a web crawler to gather smart contracts and mark the vulnerabilities through an expert. Some ML experts suggested to utilize different techniques to increase the dataset such as data augmentation.

• Different types of vulnerabilities. • Low-level and high-level vulnerabilities	Vulnerabilities captured	The types of vulnerabilities being captured is critical for any vulnerability detection system. While most of the current systems tend to focus on detecting well-known vulnerabilities, some of the interviewees suggested to explore other types of low-level vulnerabilities that are currently not being detected by existing systems and to explore detecting vulnerabilities in other blockchain platforms.
• Static code analysis • Contract bytecodes.	Integrating both source code and bytecode analysis	Most of the interviewees suggested that integrating both source code and bytecode analysis would be a better approach. They suggested to use source code analysis for detecting programming level vulnerabilities while using bytecode to detect vulnerabilities during compilation.
• Improve code analysis. • Less code errors. • Businesses losses	Proposed solution and contribution to the problem domain	The interviewees agreed that current vulnerability detection systems have limitations in terms of performance and scalability. This is because they heavily rely on expert defined rules. They also believed that the contribution to the domain will be crucial as it will prevent any financial losses caused by some of the vulnerabilities and help the overall blockchain ecosystem. Furthermore, the technological contribution to the domain involves implementing a novel approach for improving vulnerability detection in smart contracts which would aid

		the advancement of vulnerability detection systems in general.

Table 6 : Thematic analysis of interviews

2.5.3 Prototyping

Iterative prototyping involves creating, testing, and evaluating different phases of the prototype. This was essential as the author had to face numerous challenges and change certain requirements. The lack of proper standardized datasets made it difficult to get begin with. A standardized dataset had to be made with the collected smart contracts information and correct labelling had to be made.

Moreover, different data augmentation techniques and various feature engineering and selection techniques needed to be followed since certain parts of the smart contract bytecode contains unnecessary information and the author had to experiment with different algorithms that can be utilized for both source code and bytecode analysis to improve the detection results.

2.6 Summary of Findings

ID	Finding	LR	Interviews	Prototyping
1	Most of the vulnerability detection systems only focus on a few types of vulnerabilities making them less reliable and scalable.	✓		
2	Experimenting and testing different algorithms that can be utilized for vulnerability detection in smart contracts			✓
3	Identified research gaps will benefit both the domain of	✓	✓	

	Blockchain and smart contract security as well as the development of better vulnerability detection systems.			
4	Using different data imbalance handling and data generating techniques to make a large and robust dataset.	✓	✓	
5	Using a combination of both source code and bytecode analysis will help detecting vulnerabilities more accurately	✓	✓	
6	System should be able to capture not only capture the existing vulnerabilities, but also the low-level vulnerabilities		✓	
7	Using different ensemble techniques to make better predictions and reduce false detection rates.	✓	✓	
8	A necessity for a user-interface with all the working functionalities to allow users for easy access.		✓	

Table 7 : Summary of findings

2.7 Context Diagram

A context diagram is illustrated below to show an overview of the system and all its system boundaries.

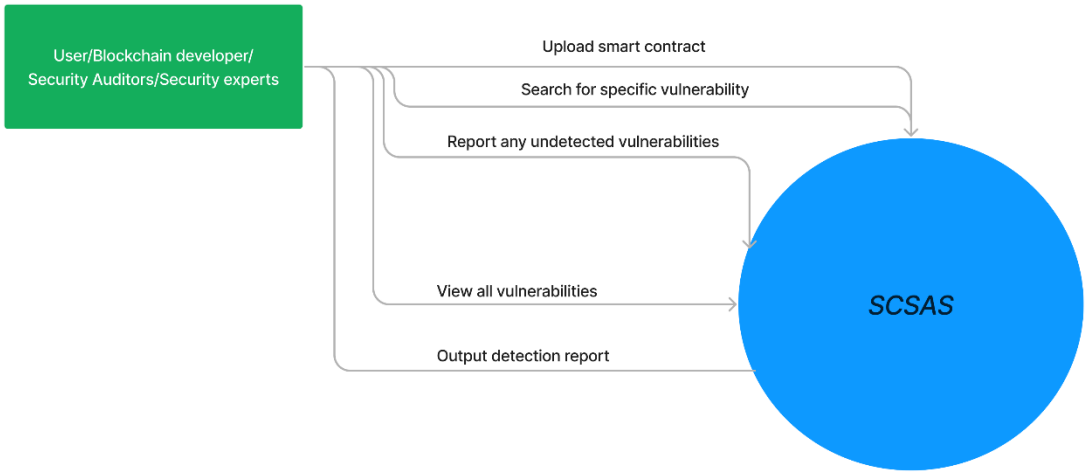

Figure 3 : Context diagram(self-composed)

2.8 Use Case Diagram

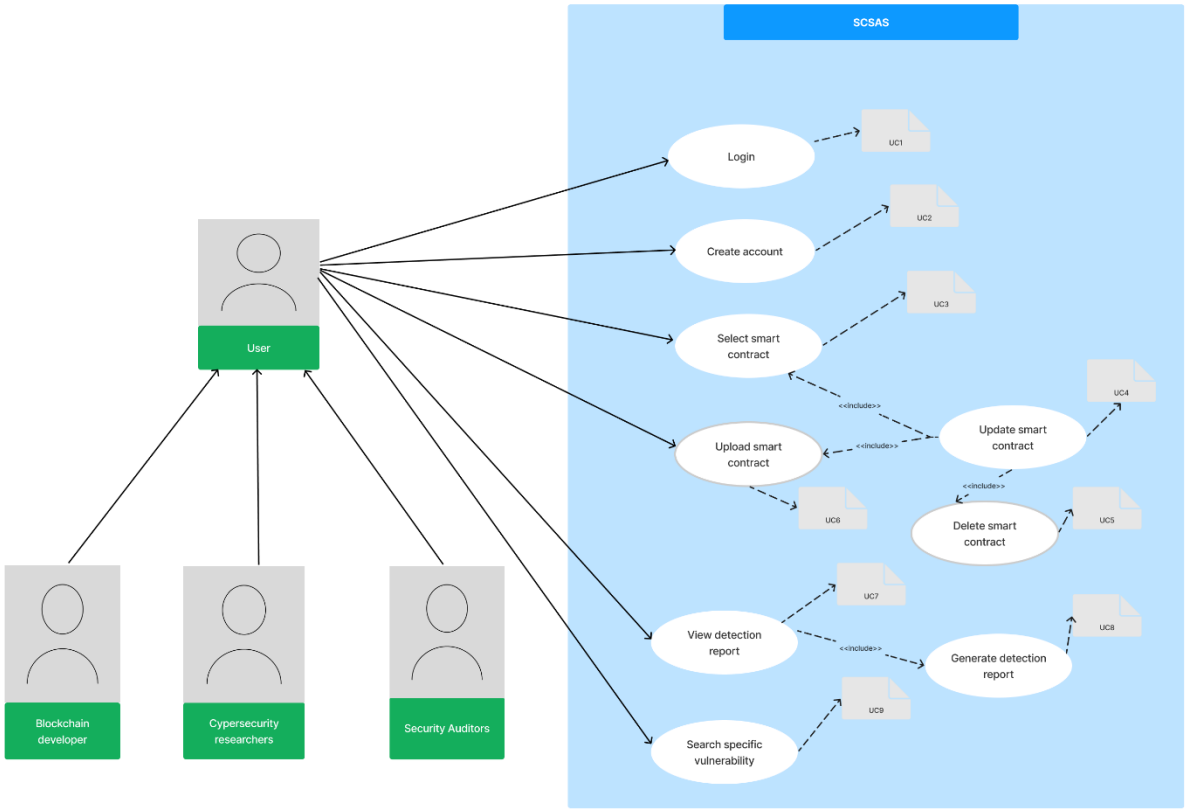

Figure 4 : Use case diagram(self-composed)

2.9 Use Case Descriptions

Only the important use case descriptions have been mentioned below.

Use case ID	UC-8	
Use case name	Generate detection report	
Use case description	A report of all the detected vulnerabilities on the smart contract sent by the user will be generated	
Actor	User	
Pre-conditions	User must upload or select an existing smart contract as input to the system	
Post-conditions	Displaying the report of all the detected vulnerabilities	
Flow of events	Actor	System

	• User will click on generate detection report	• The system will generate the detection report
Exception or error scenarios	An error message maybe prompted if a wrong address or bytecode has been uploaded into the system	
Alternative flows	None	

Table 8 : Use case description for UC-8

2.10 Requirements

The "MoSCoW" prioritization technique was utilized to prioritize the requirements for the system.

Priority Level	Description
M – must have	A mandatory(core) component needed for the prototype's functionality
S- should have	Even though they are not considered essential but, these requirements will add a lot of value.
C – could have	Considered to be optional and are not critically needed for the project.
W – will not have	The requirements that are deemed out of the project scope and not considered to be a top priority.

Table 9 : Priority levels for requirements

2.10.1 Functional Requirements

ID	Requirement	Priority level
FR-1	Users will be able to login to the system	M
FR-2	Users will be able to create an account and register themselves to the system	M
FR-3	Users must be able upload a smart contract	M
FR-5	Uploaded smart contract will be stored in a datastore	M

FR-6	Users will be able to select an existing smart contract for detection	M
FR-7	Users will be able to delete a selected/uploaded smart contract	C
FR-8	Users will be able to update a selected/uploaded smart contract	C
FR-9	Users will be able to generate a detection report of all the detected vulnerabilities	M
FR-10	Users can search for a specific type of vulnerability	M
FR-11	The system will be able to suggest corrections for the found vulnerabilities	C
FR-12	The system will be able to suggest code improvements based on best practices	C
FR-13	Users can report any unfound vulnerabilities based on the detection report	S
FR-14	Users will be able to view all the known vulnerabilities	S
FR-15	The system will show the line of code where the error is coming from	C
FR-16	System will be able to show the detection time for an uploaded/selected smart contract	C
FR-17	System will be decentralized	W

Table 10 : Functional requirements

2.10.2 Non-functional Requirements

ID	Requirement	Description	Priority level

NFR-1	Performance	The system must be able to load and identify vulnerabilities of an uploaded/selected smart contract within a few seconds with minimum resource usage. This is crucial as it will allow users to quickly identify any potential vulnerabilities and correct them.	M
NFR-2	Accuracy	Vulnerabilities must be detected accurately, since any false positives or false negatives will a significant impact on the reliability of the system.	M
NFR-2	Scalability	System must be able to detect existing and newer vulnerabilities in complex smart contracts without any significant performance drops. This will allow users to quickly upload and detect vulnerabilities in multiple smart contracts.	S
NFR-3	Usability	The system should be easily understood and useable by technical/ non-technical users .	S
NFR-4	Availability	By hosting the application on cloud platform, the system will be easily accessible for users from anywhere with a proper internet connection.	C
NFR-5	Extensibility	The system used should be easily configurable for other types of architectures and algorithms being introduced(RNN, LSTM). This will be crucial has the system might need constant improvements to increase detection results.	C
NFR-6	Generalization	Should be able to identify vulnerabilities in other programming languages(C++, Java etc.)	W

NF-7	Security	All the data should be well secured and prevent any authorized access to the system since smart contracts may contain sensitive information.	S

Table 11 : Non-functional requirements

2.11 Chapter Summary

This chapter consisted focused on the software requirements for the proposed solution. A rich picture diagram was used to illustrate all connected stakeholders for the proposed system at the start of the chapter. Moreover, an onion model of all stakeholders and their level of influence have also been presented. Finally, all requirement gathering techniques have been discussed along with defining the system requirements.

CHAPTER 03 : DESIGN

3.1 Chapter Overview

This chapter provides an overview of the design-related goals and the selected system architecture based on the mentioned goals. Moreover, system-related design choices and diagrams have also been presented with relevant justifications.

3.2 Design goals

Goals	Description
Security	Since smart contracts often deal with sensitive information such as transferring of assets or money, it is crucial that the system follows strong security measures such as the use of private keys and public keys to prevent unauthorized access.
Performance	The overall model and architecture should be designed in a way that is resource intensive and increasing its overall performance
Accuracy	One of the main aims of the system is to ensure that vulnerabilities are detected with high accuracy, reducing the number of false negatives and false positives.
Usability	The application must allow users with any level of expertise to understand the vulnerabilities been detected and the procedure with ease.
Scalability	The proposed system should be able easily adaptable with other classification models and handle smart contracts with complex functionalities in ease

Table 12 : Design goals

3.3.System Architecture

The system architecture design ensures that all the design goals are feasible and achievable. Hence, suitable architecture must be followed. A table is presented to compare various architectures and justifications are provided as to why one was selected over the other.

Architecture	Y/N	Justification
Layered architecture	N	Layered architecture allows the application to be broken down into distinct layers, where each layer is responsible for a specific part of the application. While this architecture is easier to develop and test with, it lacks scalability and is less secure. Hence layered architecture was not chosen
Tiered architecture	Y	Tiered architecture allows the application to be separated into multiple tiers, allowing for more scalability and performance. It also allows for different tiers to be secured differently, increasing the applications' overall security. These factors are closely aligned with the design goals of the application, hence tiered architecture was chosen.

Table 13 : Chosen system architecture and justification

3.3.1.Tiered Architecture

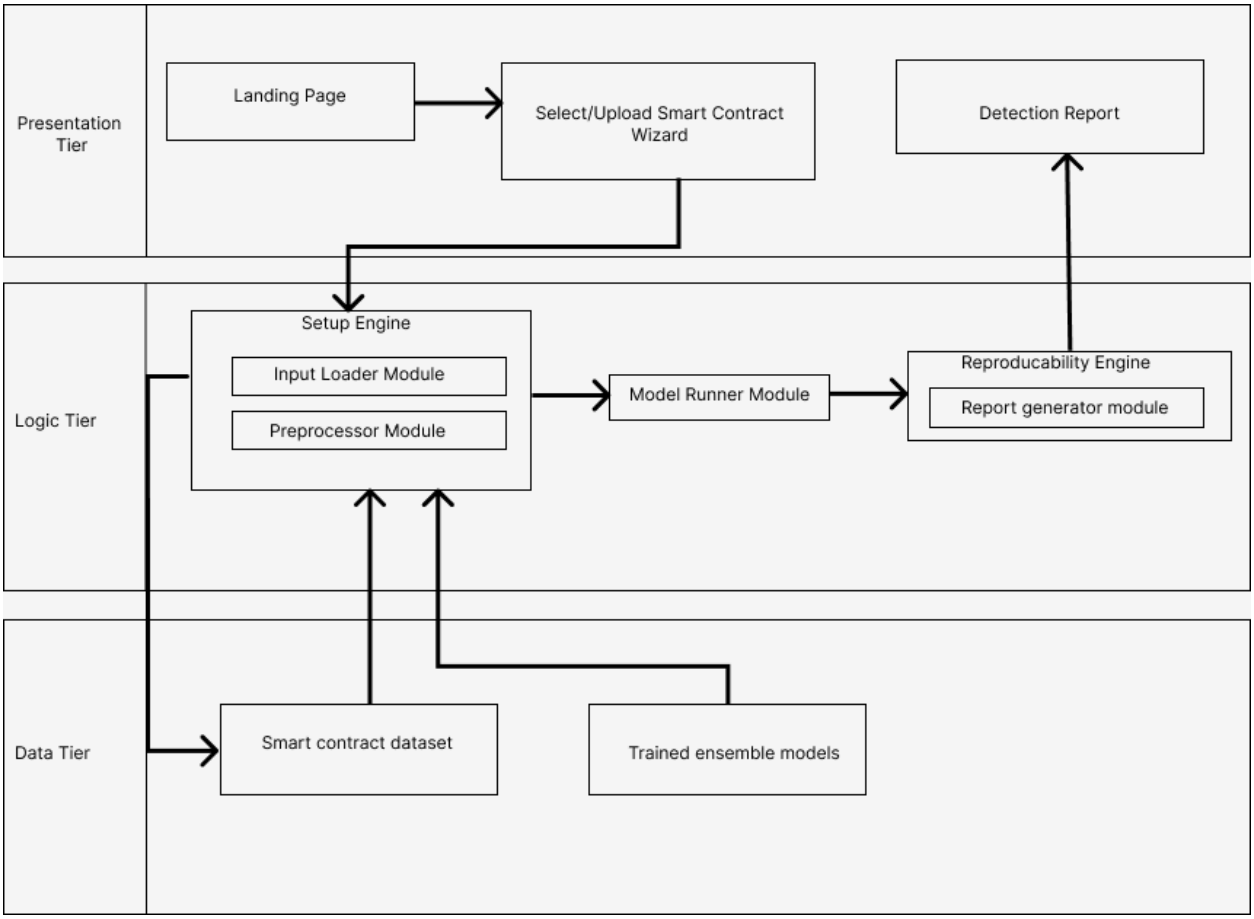

Figure 5 : Tiered architecture diagram(self-composed)

3.3.2.Discussion of the Architecture's Tiers

Data Tier –

- Smart contract dataset from trained ensemble models – All the smart contract data that has been uploaded by the user and default data have been stored.
- Trained ensemble models – The ensemble models which have been trained and will be utilized for detecting vulnerabilities.

Client Tier –

- Landing page – Initial page that the user will be directed to. It will contain an introduction to the proposed application and some of its functionalities.
- Select/Upload smart contract wizard – This page will allow users to select an existing smart contract or upload any smart contract for detection of vulnerabilities.

- Detection report – A report that will consist of all the vulnerabilities present in the smart contract that user has uploaded/selected.

Logic Tier -

- Setup Engine –
    - Input loader module – This will be utilized to load the data uploaded/selected from the user for training the model and classifying the vulnerabilities.
    - Preprocessor module – This module is responsible for handling all the preprocessing techniques from the uploaded data and converting these into inputs for the DL model.

- Model runner module – This module will be used to execute the model and get the predicted results.
- Reproducibility engine –
    - Report generator module- A report of all the detected vulnerabilities will be made from the results obtained from the predictions made by the model.

3.4.System Design

3.4.1.Design Paradigm

Design paradigms in software engineering provide different approaches for developers to design, develop and organize a solution for an identified problem. An analysis of two common design paradigms that have been considered are presented below justifications have been provided as for the selection.

Design Paradigm	Selection (Y/N)	Justification
Structured System Analysis and Design Methods(SSADM)	Y	SSADM follows a structured approach, the process is easy to implement and understand. This is crucial since the author is expected to implement and document research within 10 months . Moreover, the project is more inclined towards data

		science and some of the programming languages used are not object oriented by nature. Hence, SSADM was chosen.
Object-oriented Analysis and Design Methods(OOADM)	N	While OOADM allows for reusability of components, making it very scalable and more efficient, it is not necessary for the proposed project follows a relatively simple architecture and does not consist of any complex data handling. Hence, OOADM was not chosen

Table 14 : Choice of design paradigm and justification

3.5.Design Diagrams

3.5.1.Data Flow Diagram

The initial data flow(also known as context diagram or level 0 data flow diagram) has been shown in the SRS chapter under the context diagram. This is the data flow level 1 diagram.

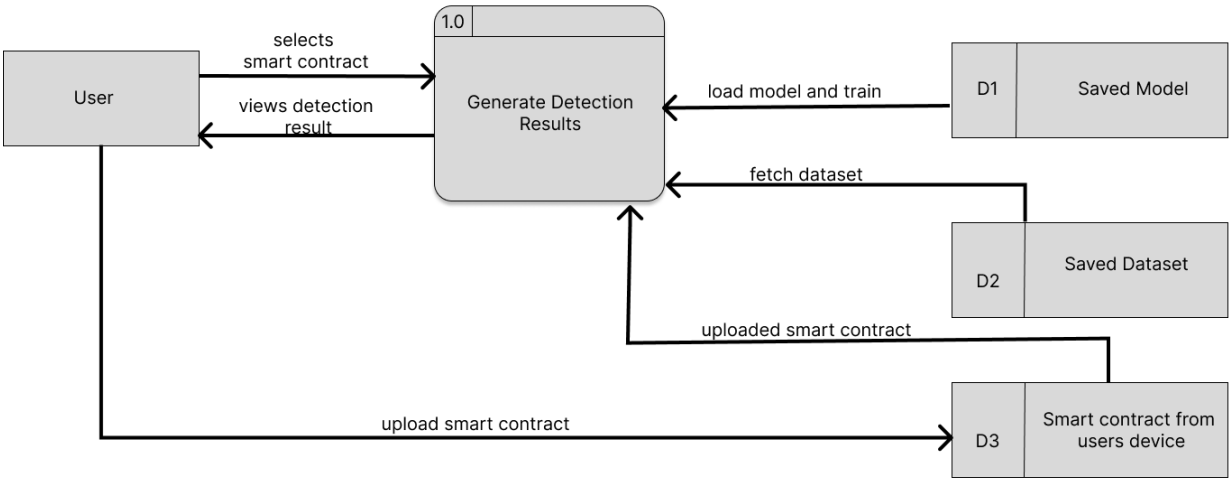

Figure 6 : Data flow level 1(self-composed)

The following diagram depicts a data flow level 2 diagram which has a detailed view of the processes involved.

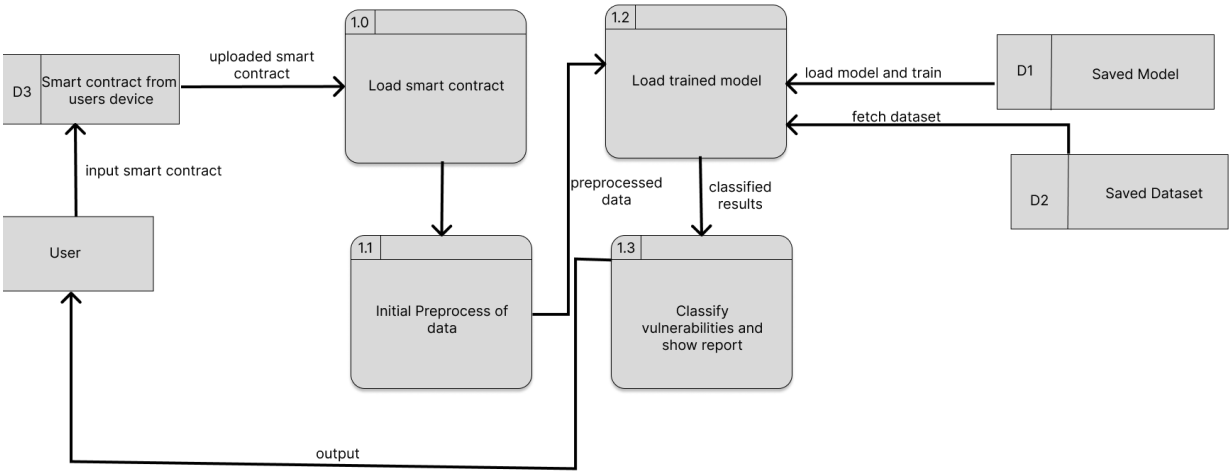

Figure 7 : Data flow level 2 diagram(self-composed)

3.5.3 Flowchart Diagram

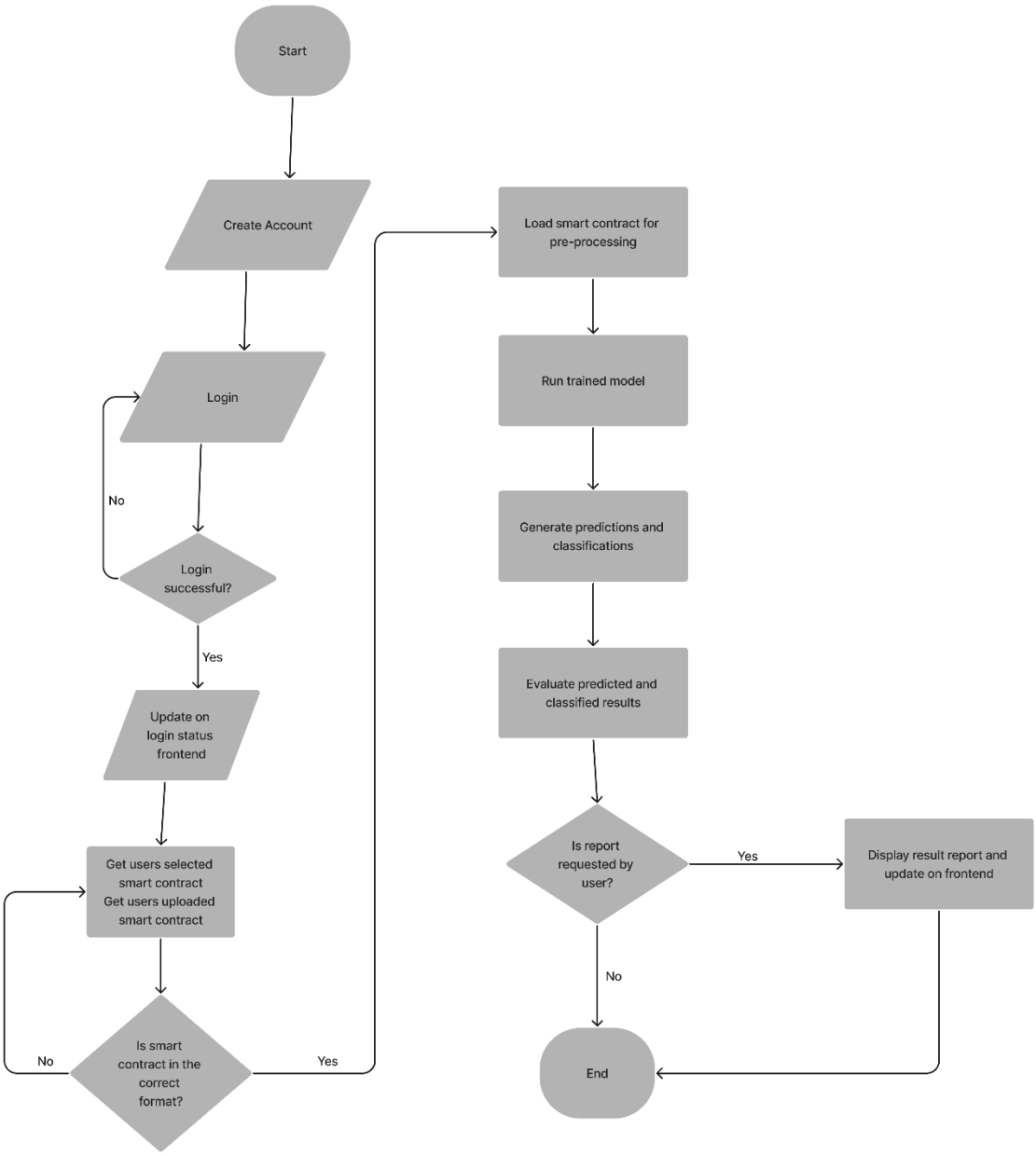

Figure 8 : System process flowchart(self-composed)

3.5.4 User-Interface Design

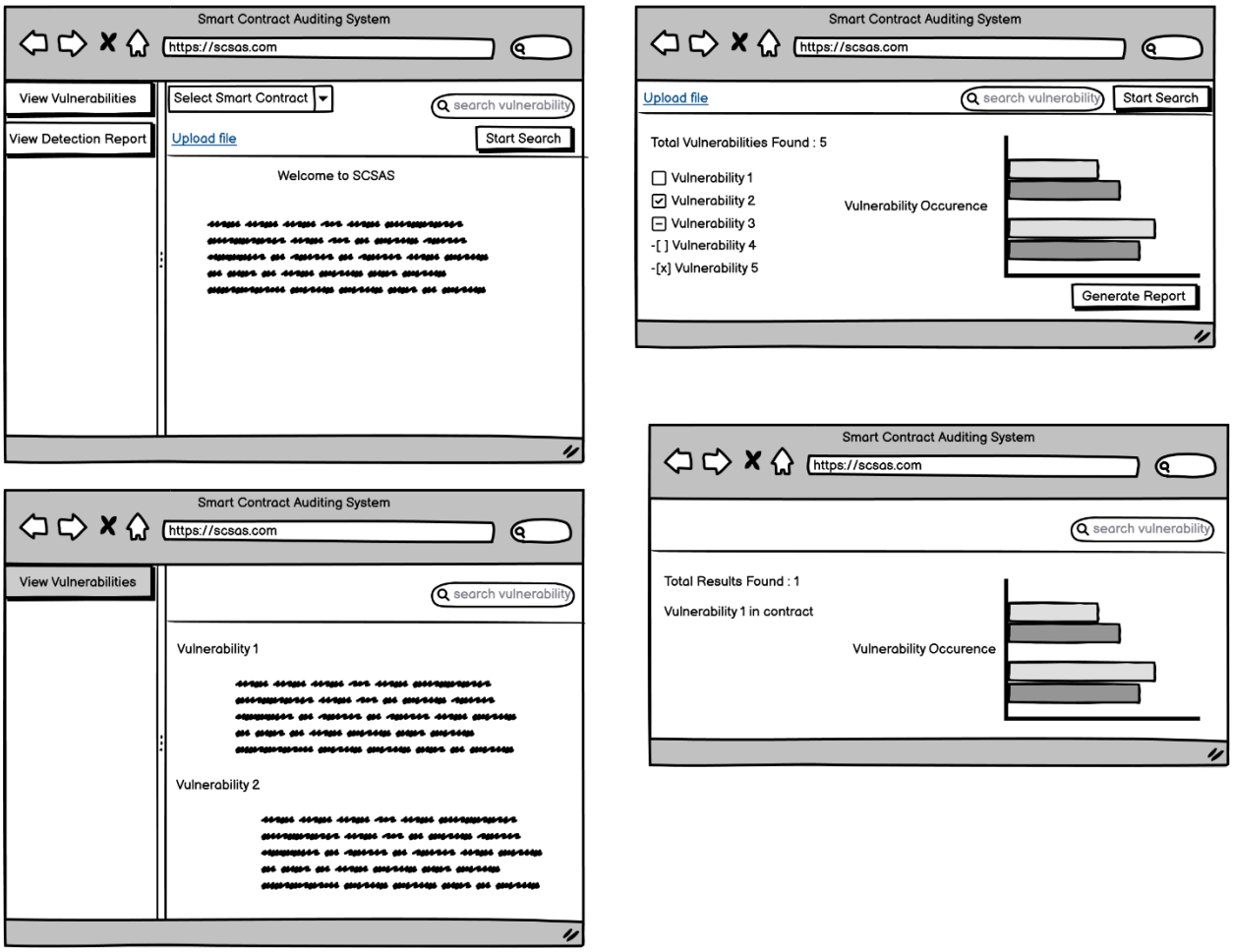

Figure 9 : UI-wireframes(self-composed)

3.5.5 User-Experience Flow

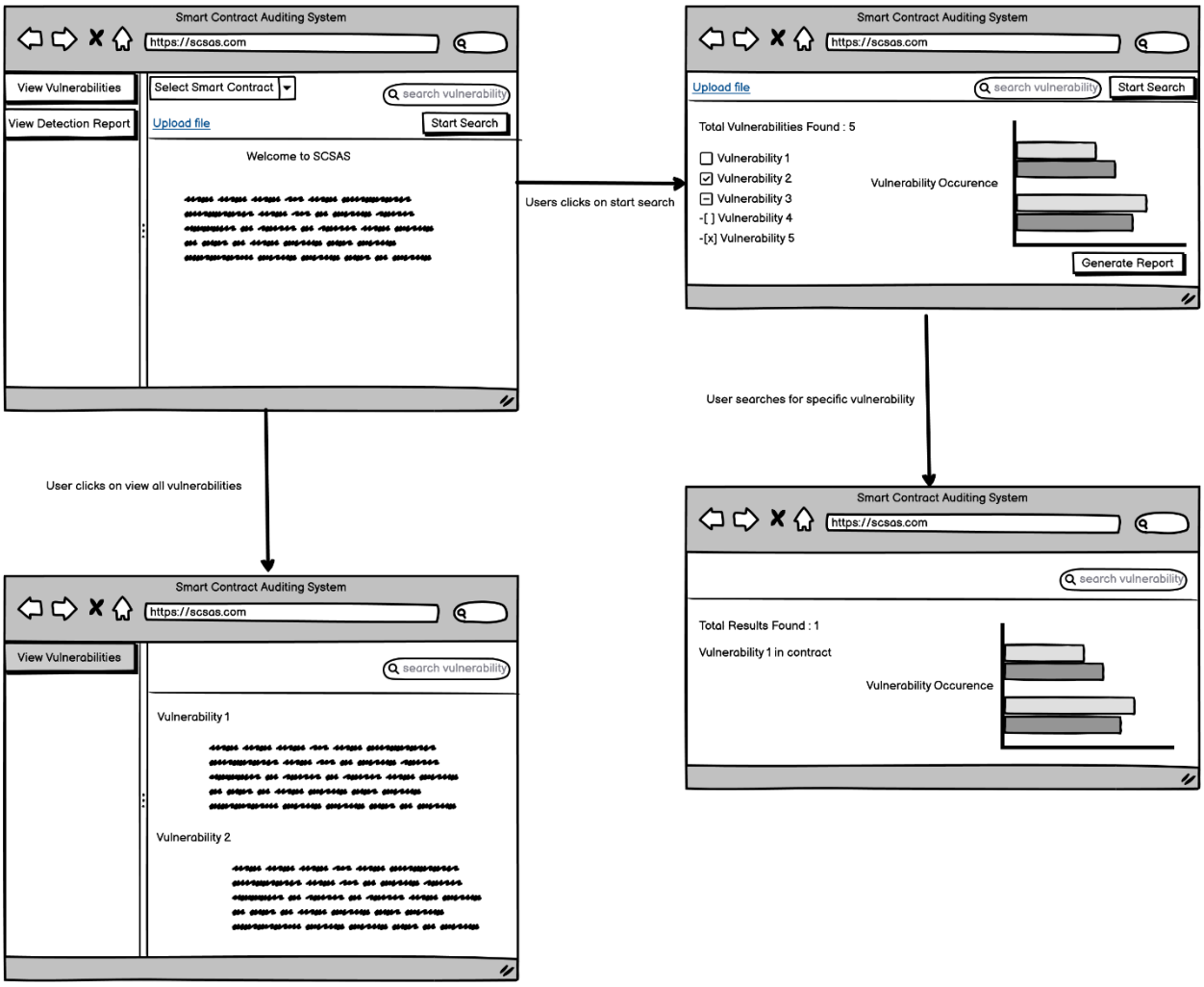

Figure 10 : User experience flow(Self-composed)

3.6 Chapter Summary

This chapter outlines all the design related goals and chosen system architecture. SSADM was chosen and all the relevant design diagrams such as the component, data flow, UI-design, activity diagrams were presented.

CHAPTER 4 : IMPLEMENTATION

4.1 Chapter Overview

This chapter will focus on the selection of appropriate tools and technologies used for the development of the project by showing a diagram of the technology stack, justifying the selections of the relevant tools. Finally, some of the code snippets used for the development of the prototype have been presented.

4.2 Technology Selection

The technology stack, dataset, programming languages, IDEs, libraries, and frameworks used for the development of the project have all been presented and justified.

4.2.1 Technology Stack

The technology stack was decided based on the requirements of the project and time, which is presented below.

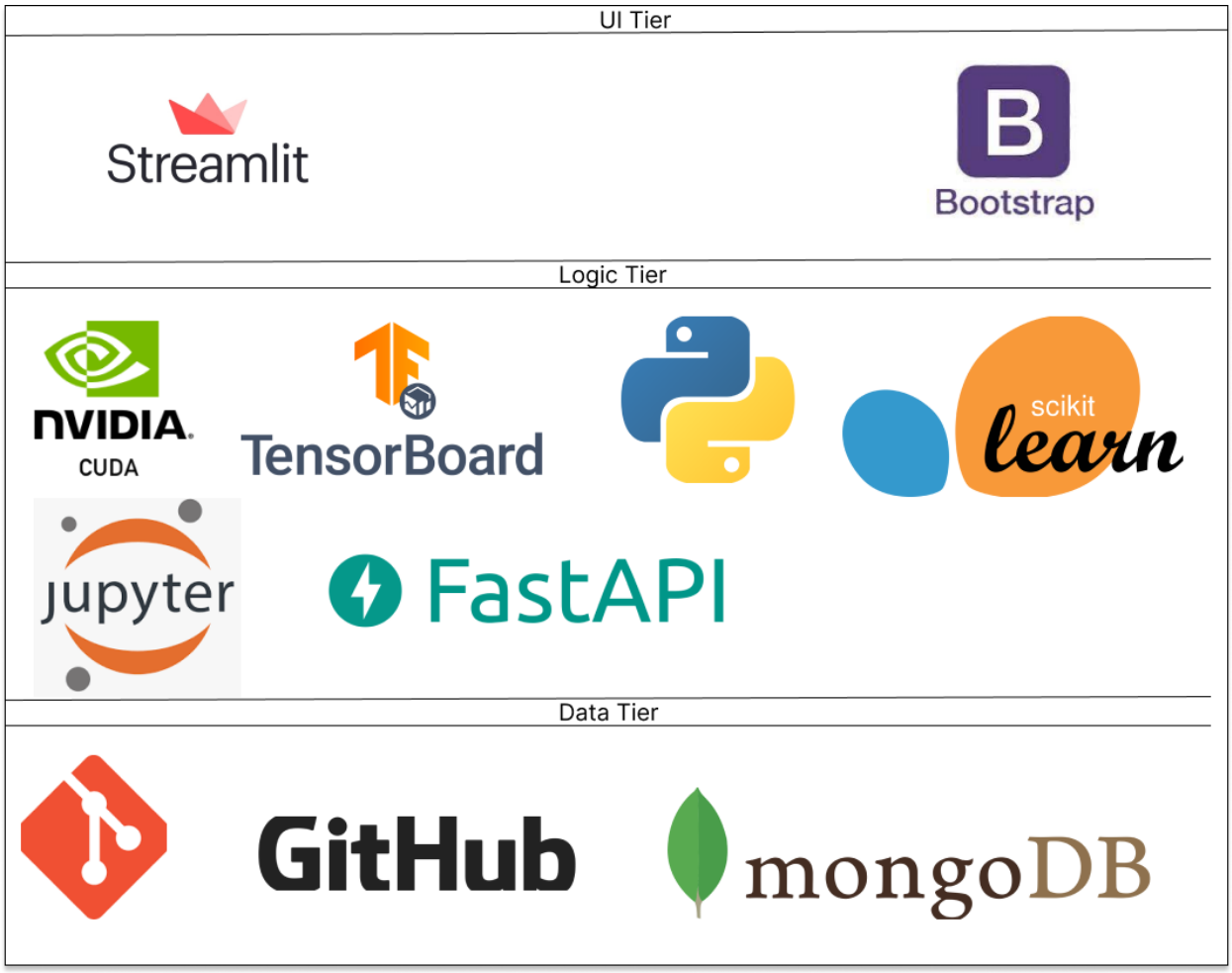

Figure 11 : Technology stack(self-composed)

4.2.2 Dataset Selection

Finding proper datasets was critical as the proposed project is based on a supervised learning approach and requires data with vulnerable and non-vulnerable code. The dataset should also be complex enough to be used for multi-class classification to detect vulnerabilities in different classes.

Name	Type	Description
ScrawID dataset	Multi-class classification	(Yashavant, Kumar and Karkare, 2022) Consists of different types of vulnerabilities from different classes that have been extracted and defined using various static analysis tools.

SmartBugs dataset	Multi-class classification	(Ferreira et al., 2020) Uses a dataset of curated from Etherscan and has been analyzed using 10 different tools, consisting of different types of vulnerabilities.

Table 15 : Dataset selection

4.2.3 Development Frameworks

The frameworks used for the project were chosen based on the authors' prior experience working with the relevant framework.

Framework	Justification
Fast API	A lightweight python-based framework. It is comparatively easy to setup, high-performing and highly customizable than other frameworks such as Django.
Bootstrap	Used for building responsive UI applications with wide range of capabilities and plugins.

Table 16 : Development framework

4.2.4 Programming Languages

Python will be used as the primary language for development as it has been used for existing projects on vulnerability detection. Python provides a wide range of libraries that can be utilized to build and test different data science projects with ease.

Moreover, it is very easy to read and understand and is widely accepted by the global community, allowing the author to develop the project with the given time constraint.

4.2.5 Libraries

The following table consists of all the required libraries that will be utilized for the project.

Library Name	Justification
Sci-kit learn	Provides a range of tools and algorithms that can be utilized for any data science related projects.

Matplotlib	Allows to perform different data visualization techniques for data exploratory analysis
Pandas	Provides various techniques for data manipulation using different data structures and tools, including tabular data which can be utilized for data analysis
Tensorflow	A widely used and supported library for building and training machine learning models.
Keras	Built as a part of tensorflow, used for building and training NN with relatively small datasets.
Streamlit	Used to build responsive web applications for data science related projects without following any complex structures.

Table 17 : Used libraries and justifications.

4.2.6 IDE

Google colab was preferred over Jupyter labs as an IDE to build and test different machine learning models since it offers more powerful resources and can collaborate with multiple devices due to its cloud-based platform environment.

Visual Studio Code was used for the development of the frontend and backend using streamlit and fast API. This IDE was due to various extensibility and dynamic features provided by Visual Studio Code.

4.2.7 Summary of technologies

The following table contains a summary of all the used technologies and used been utilized for the project.

Component	Technology Stack
Programming languages	Python,
UI framework	Bootstrap
Libraries	Pandas, Sci-kit learn, Steamlit, Matplotlib

IDEs	Google colab, Visual studio code
Frameworks	Tensorflow, Keras
Version control	Git, GitHub

Table 18 : Summary of all the components and technology stack

4.3 Implementation of Core functionalities

Since the dataset consists of smart contract bytecodes, they had to be transformed into their following opcode instructions. These instructions will be used to identify which specific opcode value has a vulnerability.

After getting the opcodes for each bytecode, they had to be transformed into proper inputs to train the NN which involved heavy preprocessing.

```
def contract_preprocess(df):
    common_words = 1000
    max_len = 130

    # Class Tokenizer - This class allows to vectorize a text corpus
    tokenizer = Tokenizer(num_words=common_words, lower=False)

    tokenizer.fit_on_texts(df['OPCODE'].values)
    sequences = tokenizer.texts_to_sequences(df['OPCODE'].values)
    word_index = tokenizer.word_index
    print('Found %s unique tokens.' % len(word_index))
    X = pad_sequences(sequences, maxlen=max_len)
    return X
contract_preprocess(data)
```

Figure 12 : Preprocessing code segment(self-composed)

A tokenizer class was used to convert all the opcodes into a series of tokens which is then used as inputs to the NN .

After converting the opcodes into tokens. The vulnerabilities that had been detected had to be labelled. This was done by marking the types of vulnerability category each opcode went into. In this way, categories can be used to add more types of vulnerabilities according to their respective opcode sequence.

```
Non-v = shuffled[shuffled['CATEGORY'] == '1 0 0 0'] # no vulnerabilities
v1 = shuffled[shuffled['CATEGORY'] == '0 1 0 0'] # v1
v2 = shuffled[shuffled['CATEGORY'] == '0 0 1 0'] # v2
v3 = shuffled[shuffled['CATEGORY'] == '0 0 0 1'] # v3
v4 = shuffled[shuffled['CATEGORY'] == '0 1 1 0'] # v4
```

Figure 13 : setting vulnerabilities code segment(self-composed)

Finally, the data had to be split into training, testing and validating and then use the training data to train the NN. The code below shows the training of the proposed CNN model with slight optimizations.

```
n_most_common_words = 1000 #150
model = Sequential()
model.add(Embedding(n_most_common_words, 128, input_length=X_train_res.shape[1]))
model.add(Conv1D(filters=64, kernel_size=5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(units=128, activation='relu'))
model.add(Dropout(rate=0.5))
model.add((Dense(2, activation='sigmoid')))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
print(model.summary())
start_time = time.time()
history = model.fit(X_train_res, ytrainres_cat, epochs=50, batch_size=256 ,
                    validation_split=0.0, validation_data=(X_val_res, yvalres_cat))

end_time = time.time()
print('Time taken for training: ', end_time-start_time)
```

Figure 14 : Trained model(self-composed)

4.4 Chapter Summary

This chapter provided all the required technologies and tools for the development of the research project. Moreover, relevant code snippets of the initial implementation have been shown with relevant explanations for each of them.

CHAPTER 5 : CONCLUSION

5.1 Chapter Overview

This chapter provides an overview of the project related deviations that had to occur in terms of the scope and schedule. A roadmap of all the requirements that have been achieved and what to be completed is also presented, alongside some of the initial test results from the prototype and some of the improvements have been presented below.

5.2 Deviations

5.2.1 Scope related deviations.

The implementation of the proposed solution had a few deviations from the initial project proposal. Different types of ensemble techniques had to be tried in order to find the best performing model and sometimes, using ensemble techniques would not always guarantee good accuracy. This will further be evaluated and tested with different feature selection techniques to find out which model can give the best accuracy when tested consistently.

5.2.2 Schedule related deviations

There were no schedule related deviations that had occurred from the Gantt chart. All the scheduled deliverables were made on time.

5.3 Initial Test Results

The author had tested with 3 different models for now with the chosen dataset and used both testing and validation accuracy. These models have not been properly optimized and the author has used baseline models.

The LSTM model was implemented and tested which gave an accuracy of around 71%.

```
Epoch 13/20
44/44 [==============================] - 28s 626ms/step - loss: 0.0743 - acc: 0.9749 - val_loss: 0.8413 - val_acc: 0.7562
Epoch 14/20
44/44 [==============================] - 27s 609ms/step - loss: 0.0702 - acc: 0.9771 - val_loss: 0.9110 - val_acc: 0.7482
Epoch 15/20
44/44 [==============================] - 27s 614ms/step - loss: 0.0642 - acc: 0.9797 - val_loss: 0.7734 - val_acc: 0.8111
Epoch 16/20
44/44 [==============================] - 27s 619ms/step - loss: 0.0623 - acc: 0.9816 - val_loss: 0.9858 - val_acc: 0.7395
Epoch 17/20
44/44 [==============================] - 26s 580ms/step - loss: 0.0536 - acc: 0.9840 - val_loss: 1.1804 - val_acc: 0.7225
Epoch 18/20
44/44 [==============================] - 25s 564ms/step - loss: 0.0610 - acc: 0.9820 - val_loss: 0.9809 - val_acc: 0.7250
Epoch 19/20
44/44 [==============================] - 25s 575ms/step - loss: 0.0524 - acc: 0.9849 - val_loss: 1.3486 - val_acc: 0.7171
Epoch 20/20
44/44 [==============================] - 29s 669ms/step - loss: 0.0529 - acc: 0.9846 - val_loss: 0.9689 - val_acc: 0.7211
Time taken for training:  531.0028395652771
```

Figure 15 : LSTM model(self-composed)

A baseline CNN model was implemented and tested and got a validation accuracy of around 65%, which is relatively low.

```
Epoch 43/50
44/44 [==============================] - 3s 73ms/step - loss: 4.1254e-05 - acc: 1.0000 - val_loss: 1.3036 - val_acc: 0.6556
Epoch 44/50
44/44 [==============================] - 3s 76ms/step - loss: 4.7360e-05 - acc: 1.0000 - val_loss: 1.2734 - val_acc: 0.6592
Epoch 45/50
44/44 [==============================] - 3s 74ms/step - loss: 4.7307e-05 - acc: 1.0000 - val_loss: 1.4215 - val_acc: 0.6501
Epoch 46/50
44/44 [==============================] - 3s 76ms/step - loss: 4.9679e-05 - acc: 1.0000 - val_loss: 1.4273 - val_acc: 0.6505
Epoch 47/50
44/44 [==============================] - 3s 76ms/step - loss: 4.5611e-05 - acc: 1.0000 - val_loss: 1.3693 - val_acc: 0.6534
Epoch 48/50
44/44 [==============================] - 3s 75ms/step - loss: 5.0900e-05 - acc: 1.0000 - val_loss: 1.3037 - val_acc: 0.6585
Epoch 49/50
44/44 [==============================] - 3s 73ms/step - loss: 4.8521e-05 - acc: 1.0000 - val_loss: 1.4672 - val_acc: 0.6505
Epoch 50/50
44/44 [==============================] - 3s 69ms/step - loss: 4.1856e-05 - acc: 1.0000 - val_loss: 1.4255 - val_acc: 0.6520
```

Figure 16 : CNN model(self-composed)

5.4 Required Improvements

Although the core research component had been developed and tested, the author would make some crucial improvements to the proposed solution within the given time frame.

- Improve the accuracy, precision , recall and f1-score of the proposed model.
- Make the architecture more scalable by experimenting with different techniques such as transfer learning.
- Balance the dataset using different techniques such as SMOTE or even GANs.

5.5 Demo Video

A video demonstrating the initial implementation has been uploaded to YouTube and can be viewed here : https://youtu.be/F5AJg22SWdo

5.6 Chapter Summary

This concluding chapter provides an overview of all the scope and schedule related deviations that had occurred during the initial implementation phase of the project. The author has also provided all the further improvements and some of the initial test results alongside a demonstration of the proposed solution.

REFERENCES

Gopali, S. et al. (2022). Vulnerability Detection in Smart Contracts Using Deep Learning. 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC). June 2022. 1249–1255. Available from https://doi.org/10.1109/COMPSAC54236.2022.00197.

Tikhomirov, S. et al. (2018). SmartCheck: Static Analysis of Ethereum Smart Contracts. 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). May 2018. 9–16. Ye, J. et al. (2022). Vulpedia: Detecting vulnerable ethereum smart contracts via abstracted vulnerability signatures. Journal of Systems and Software, 192, 111410. Available from https://doi.org/10.1016/j.jss.2022.111410 [Accessed 8th November 2022].

Li, Z. et al. (2019). A Comparative Study of Deep Learning-Based Vulnerability Detection System. IEEE Access, 7, 103184–103197. Available from https://doi.org/10.1109/ACCESS.2019.2930578

Palladino, S. (2017). The Parity Wallet Hack Explained. OpenZeppelin blog. Available from https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/ [Accessed 30th October 2022].

Gao, Z. (2020). When Deep Learning Meets Smart Contracts. 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). September 2020. 1400–1402. Zhuang, Y. et al. (2020). Smart Contract Vulnerability Detection using Graph Neural Network. 9th July 2020. 3283–3290. Available from https://doi.org/10.24963/ijcai.2020/454 [Accessed 14th October 2022].

Yu, X. et al. (2021). DeeSCVHunter: A Deep Learning-Based Framework for Smart Contract Vulnerability Detection. 2021 International Joint Conference on Neural Networks (IJCNN). July 2021. 1–8. Available from https://doi.org/10.1109/IJCNN52387.2021.9534324.

Gopali, S. et al. (2022). Vulnerability Detection in Smart Contracts Using Deep Learning. 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC). June 2022. 1249–1255. Available from https://doi.org/10.1109/COMPSAC54236.2022.00197.

Lutz, O. et al. (2021). ESCORT: Ethereum Smart COntRacTs Vulnerability Detection using Deep Neural Network and Transfer Learning. undefined. Available from https://www.semanticscholar.org/reader/1c5e67742dab39626829c1cbb099cd9738c26ee8 [Accessed 24th October 2022].

Zhang, L. et al. (2022). SPCBIG-EC: A Robust Serial Hybrid Model for Smart Contract Vulnerability Detection. Sensors, 22 (12), 4621. Available from https://doi.org/10.3390/s22124621 [Accessed 23rd October 2022].

Qian, P. et al. (2022). Smart Contract Vulnerability Detection Technique: A Survey. Available from https://doi.org/10.13328/j.cnki.jos.006375 [Accessed 13th October 2022].

Kushwaha, S.S. et al. (2022). Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract. IEEE Access, 10, 6605–6621. Available from https://doi.org/10.1109/ACCESS.2021.3140091.

Tantikul, P. and Ngamsuriyaroj, S. (2020). Exploring Vulnerabilities in Solidity Smart Contract: Proceedings of the 6th International Conference on Information Systems Security and Privacy. 2020. Valletta, Malta: SCITEPRESS - Science and Technology Publications, 317–324. Available from https://doi.org/10.5220/0008909803170324 [Accessed 26th October 2022].

Zhang, L. et al. (2022). SPCBIG-EC: A Robust Serial Hybrid Model for Smart Contract Vulnerability Detection. Sensors, 22 (12), 4621. Available from https://doi.org/10.3390/s22124621 [Accessed 23rd October 2022]. Lin, G. et al. (2020). Software Vulnerability Detection Using Deep Neural Networks: A Survey. Proceedings of the IEEE, 108 (10), 1825–1848. Available from https://doi.org/10.1109/JPROC.2020.2993293. Introduction to smart contracts. (2022). ethereum.org. Available from https://ethereum.org [Accessed 24th October 2022].

Praitheeshan, P. et al. (2020). Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey. Available from https://doi.org/10.48550/arXiv.1908.08605 [Accessed 14th October 2022].

Rameder, H., di Angelo, M. and Salzer, G. (2022). Review of Automated Vulnerability Analysis of Smart Contracts on Ethereum. Frontiers in Blockchain, 5, 814977. Available from https://doi.org/10.3389/fbloc.2022.814977 [Accessed 29th October 2022].