INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER

SHERLOCK: A Deep Learning Approach To Detect Software Vulnerabilities

A Thesis by

Mr. Saadh Jawwadh

Supervised by

Mr. Guhanathan Poravi

Submitted in partial fulfilment of the requirements for the BSc in Computer Science degree at the University of Westminster.

May 2023

Abstract

The increasing reliance on software in various applications has made the problem of software vulnerability detection more critical. Software vulnerabilities can lead to security breaches, data theft, and other negative outcomes. Traditional software vulnerability detection techniques, such as static and dynamic analysis, have been shown to be ineffective at detecting multiple vulnerabilities.

To address this issue, this study employed a deep learning approach, specifically Convolutional Neural Networks (CNN), to solve the software vulnerability detection problem. A 5-split cross-validation approach was used to train and evaluate the CNN model, which takes tokenized source code as input.

The findings indicated that Sherlock successfully detected multiple vulnerabilities at the function level, and its performance was particularly strong for CWE-199, CWE-120, and CWE-Other, with an overall high accuracy rate and significant true positive and true negative values. However, the performance was less reliable for some vulnerabilities due to the lack of a standardized dataset which will be a future research direction. The results suggest that compared to current techniques, the proposed deep learning approach has the potential to substantially enhance the accuracy of software vulnerability detection.

Keywords: Software Vulnerability Detection, AI, Deep Learning, Convolutional Neural Network, Gaussian Noise.

Subject Descriptors:

- Security and privacy → Systems security → Vulnerability management → Vulnerability scanners
- Computing methodologies → Artificial intelligence
- Computing methodologies → Machine learning → Machine learning approaches → Neural networks

Declaration

I hereby declare that this project report and all the artifacts associated with it is my own work, and it has not been submitted before and it is currently being submitted for any degree program.

Name of Student: Saadh Jawwadh
Registration No: 2019175 / w1790792

Signature: Saadh Jawwadh (Signed) Date: 2023/05/10

Acknowledgment

"**Thank you**" is a simple phrase, but it cannot fully express the depth of my gratitude. To me, this phrase symbolizes the fact that I could not have made it this far without your valuable time, guidance, support, motivation, tolerance, kindness, and everything else you have provided for me. I truly appreciate, admire, and will never forget what you have done for me.

Thank you, Mr. Guhanathan Poravi (supervisor), you were the shepherd who created the path for the research, led with examples, reviewed every part of the work, pushed beyond limits. I admire your dedication to how you made your entire life around lecturing and mentoring. I know you're not a follower of religion yet, I pray for you cause that's the ultimate gratitude I can offer for you other than your favorite cheesecakes.

Thank you, you are the reason for everything that I was, am, and will be. Denzel Washington once said, "A parent is the first love of a child, but the child is the last love of a parent." You both have sacrificed so much for me, and I want to assure you that I will do the same for you. The rest of my gratitude will be actions.

Thank you, my gratitude to the evaluators, experts, seniors, lecturers, and friends for their invaluable time, feedback, expertise, and willingness. Your contributions have been instrumental in shaping my work and helping me to improve.

Thank you, Farhana Aunty (neighbor), Aaru amma and appa, Visha amma and appa (parents of friends), for your warm hospitality, which I truly appreciate. Your kindness means a lot to me, and even though you may never read this, it lightens my heart to acknowledge each and every one of you.

Contents

List of figures

List of tables

Acronym	**Description**
AI	Artificial Intelligence
CNN	Convolutional Neural Network
CWE	Common Weakness Enumeration
TN	True Negative
FN	False Negative
TP	True Positive
FP	False Positive
GUI	Graphical User Interface
HDF5	Hierarchical Data Format version 5
ML	Machine Learning

NLTK	Natural Language Toolkit
SQL	Structured Query Language
OSF	Open Science Framework
TF	TensorFlow
UI	User Interface
SLEP	Social, Legal, Ethical and Professional

CHAPTER 1: INTRODUCTION

1.1 Chapter overview

This chapter will provide an overview of the rise of software vulnerabilities, analyze current detection methodologies, and establish the need for research on the problem. It will also cover the author's aim, research motivation, and how difficult the research will be to contribute to a novel solution.

1.1.1 Software vulnerabilities

Software vulnerability can be defined as “A security flaw, glitch, or weakness found in software code that could be exploited by an attacker (threat source) ” (Dempsey *et al.*, 2019, p. 105). The rapid expansion of computer systems and interconnected software has also resulted in an increase in software vulnerabilities. For example, a 2017 software failure resulted in 1.7 million USD in financial loss and 268 years of downtime (Hanif et al., 2021). Furthermore, companies are spending more than 82 per cent more on cyber security in 2021 than they did in 2020. (State of Cybersecurity Report 2021 | 4th Annual Report | Accenture, 2021) Such a dramatic increase in investment, interest and software vulnerabilities necessitates a concern for state-of-the-art detection methods.

1.1.2 Vulnerability detection

Current well-known solutions such as firewalls, user authentication, access control, and cryptography are inadequate for today's cyber security requirements (Sarker et al., 2020) and detecting vulnerabilities after the software has been deployed is ineffective. Furthermore, the ideal solution would detect vulnerabilities prior to deployment. The rise in vulnerabilities found after software products were released indicates that state-of-the-art vulnerability detection techniques need to be improved to a more efficient and effective state (Lin et al., 2020).

Current detection techniques can be divided into static, dynamic, and hybrid approaches. Static approaches include techniques such as rule-based analysis, code similarity detection, and symbolic execution, which rely on source code analysis and have a significantly higher false positive rate, making those approaches unreliable (Pewny et al., 2014; Lin et al., 2020). Dynamic approaches

include techniques such as fuzzy testing and taint analysis which has been addressed as having low code coverage issues (Newsome and Song, 2005; Portokalidis, Slowinska and Bos, 2006; Lin et al., 2020). As a result, hybrid approaches emerged to address the issues raised by both static and dynamic approaches however, they also seem to have their own limitations and do not lessen the current software vulnerabilities (Yamaguchi et al., 2014; Lin et al., 2020).

1.1.3 Artificial intelligence-based muti vulnerability detection.

Since the existing technique proved ineffective, a new approach was investigated, and a data-driven machine learning-based approach has shown promising results in detecting vulnerabilities (Sun et al., 2019; Coulter et al., 2020; Lin et al., 2020). Despite the fact that artificial intelligence is well-researched in other domains, there is significantly low research with proof of concepts that have been addressed in software vulnerability detection. (Ghaffarian and Shahriari, 2018; Sun et al., 2019) Therefore, it is evident that there is a huge need for research in artificial intelligence-based software vulnerability detection.

1.2 Problem definition

Software vulnerabilities have become a widespread issue for the modern generation, and exploitable vulnerabilities can pose a threat to computer systems (Lin et al., 2020; Heartbleed Bug, 2020). Although the current solution can detect single vulnerabilities after software deployment, an effective solution should detect vulnerabilities prior to software deployment (Lin et al., 2020). Despite the fact that current solutions rely on unsupervised learning-based models, it is suggested and demonstrated through a successful case study that using deep learning approaches will improve the ability to detect known and unknown vulnerabilities (Lin et al., 2020; Sonnekalb, Heinze and Mäder, 2021).

1.3 Problem statement

Detecting software vulnerabilities after deployment is equivalent to patching a hole in a bucket full of water because the damage has already been done; therefore, detecting multiple software vulnerabilities prior to deployment is critical.

1.4 Research motivation

Software vulnerabilities are becoming a more widespread problem as we increasingly rely on software components to complete our tasks. This also increases the risk of becoming a victim of software vulnerabilities. As stated in the problem domain, the author's primary impetus for researching this domain was the rapid growth of vulnerabilities (Ryan, 2022).

In addition, a victim case study involving the author's friend who lost almost all of the crucial documents they had on their laptop as a result of an attack led the author to learn about software vulnerabilities and spark an interest in the topic.

1.5 Research gap

Previous literature mentioned that there is a significant need for an unsupervised approach to identifying vulnerabilities. Existing solutions are based on a supervised approach which can only detect known vulnerabilities and current solutions are limited to single vulnerability detection. Further, existing multi-vulnerability detection solutions are still on the conceptual level (Hanif et al., 2021; Li et al., 2022; Singh, Grover and Kumar, 2022).

Existing detection methods are missing to address various vulnerabilities which can cause serious problems after the deployment of the software (Li et al., 2022). A fine-tuned deep-learning solution that can overcome this problem and as be identified in this project.

1.6 Novelty of the research

1.6.1 Problem novelty

Software vulnerability detection is not a new problem; it has been a research concern for a long time. Despite the fact that AI advancement has created a new variation of the problem, software vulnerability detection with AI is less researched and also shows promising results which show using AI to detect software vulnerabilities can be a novel problem to solve (Ghaffarian and Shahriari, 2018; Li et al., 2022).

1.6.2 Solution novelty

The existing solutions for detecting software vulnerabilities are based on static and dynamic detection (Pewny et al., 2014; Lin et al., 2020). Even though there are researchers who analyzed

artificial intelligence-based approaches most of them are conceptual frameworks.(Sun et al., 2019; Coulter et al., 2020; Lin et al., 2020) . As a result, the vacuum for an artificial intelligence-based solution has yet to be filled. This will pave the path for a novel software vulnerability detection solution with AI which is also proposed by the author.

1.7 Contribution to the body of knowledge

The anticipated contributions to the body of knowledge will be made by offering a novel supervised learning-based model to identify multiple vulnerabilities before the deployment of software.

1.7.1 Contribution to the problem domain

It has been proposed that detecting multiple vulnerabilities before the software deployment would be the ideal solution (Lin et al., 2020). Whereas the currently available solutions are primarily focused on detecting single vulnerabilities after the state of deployment. Therefore, it is hypothesized to provide an ideal solution which can detect software vulnerabilities prior to deployment.

1.7.2 Contribution to the research domain

It has been proposed to use a brand-new ensembled model based on unsupervised learning to identify software vulnerabilities through the semantics of the code. Since existing solutions rely on models of unsupervised learning. Therefore, it is proposed to provide an artificial intelligence-based solution which can detect multiple vulnerabilities in an effective way.

1.8 Research challenge

Although there is a substantial body of prior research on software vulnerability identification, the majority of those publications are conceptual frameworks and related unsupervised learning techniques (Sonnekalb, Heinze and Mäder, 2021).

Even though a supervised model can considerably improve sensitivity for detecting multiple and unique vulnerabilities, deploying such a model can be difficult due to a lack of labelled and synthetic datasets (Sonnekalb, Heinze and Mäder, 2021; Li et al., 2022).

Prior works on supervised approaches are significantly lower compared to other domains, (Sarker et al., 2020) which will pose a challenge because there are no known paths to follow in this specific domain, which may result in a dead end.

Because of the ever-increasing nature of software vulnerabilities, it was impossible to overcome them with a single solution and required staying current on less known and zero-day vulnerabilities. As a result, staying up to date on new case studies will be a major challenge.

Given the foregoing factors, it will be difficult to develop a prototype that can detect software vulnerabilities using artificial intelligence as the project addresses the aforementioned challenges.

1.9 Research questions

RQ1 -What are the components required to develop an artificial intelligence model capable of detecting multiple vulnerabilities prior to the software deployment?

RQ 2 – What are the recent advancements in software vulnerability detection?

RQ3 – How to develop a novel artificial intelligence model capable of detecting multiple software vulnerabilities prior to software deployment?

1.10 Research aim

The aim of the research is to design, develop and evaluate a novel software vulnerability detection system with artificial intelligence.

In a nutshell, the author's goal is to develop an artificial intelligence-based system that can identify multiple software vulnerabilities before the software is released, using the right data and technology combinations, which should greatly enhance vulnerability detection when compared to current approaches.

Prior to the proceeding, the required knowledge of the domain and technologies throughout the research is intended to be obtained in various ways. Additionally, the system was designed to be hosted and made available to the public while also being open source in a controlled manner.

1.11 Research objective

Objectives	Description	Learning outcome	Research Questions
Problem Identification	**RO1**: Conducting thorough prior research on software vulnerabilities to gain a comprehensive understanding of the problem. **RO2:** Identifying and understanding current techniques used to solve the problem.	LO1, LO2	RQ1, RQ2
Literature Review	**RO3**: Learn about and comprehend existing software vulnerability detection solutions and prior works. **RO4**: Compare and contrast existing techniques in software vulnerability detection. **RO5**: Discover the limitations of existing software vulnerability detection solutions. **RO6:** Forecast the difficulties to be overcome based on previous work.	LO1, LO4, LO8	RQ1, RQ2, RQ3
Methodology Selection and SLEP Framework	**RO7:** Find and employ the best research methodology for the project. **RO8:** Find and employ the best development methodology for the research. **RO9:** Find and employ the best design methodology for the research.	LO2, LO6, LO8	RQ1, RQ3

	R1O: Find and employ the best solution methodology for the research. **R11:** Find and employ the best testing methodology for the research. **RO12:** Forecast and prioritize risks and develop a risk-mitigation strategy. **RO13:** Prioritize and schedule tasks. **RO14:** Recognize and contrast social, legal, ethical, and professional issues.		
Requirement Elicitation	**RO15:** Designing and carrying out surveys and interviews. **RO16:** Through the survey, gather requirements and software vulnerability detection-related insights. **RO17:** Through interviews, gather requirements and additional insights from domain experts. **RO18:** Determine and map potential use cases. **RO19:** Understand and finalize stakeholder and usability requirements.	LO1, LO3, LO5, LO8	RQ1
Design	**RO20:** Pose a high-level architecture to complement the design. **RO21:** Clearly define the design goals. **RO22:** Pose a data flow diagram and better understand the data flow in the research.	LO1, LO5, LO8	RQ1, RQ3

	RO23: Picture and design a model to detect software vulnerabilities using AI		
Implementation	**RO22:** Look up and select the appropriate technologies and tools for implementation. **RO23:** Develop a novel model to detect multiple software vulnerabilities. **RO24:** Develop a user-friendly interface for the model and integrate it with core functionalities.	LO1, LO5, LO7, LO8	RQ3
Evaluation	**RO25:** Create appropriate test plans for both the model and the user interface based on the chosen test methodology and use cases. **RO26:** Evaluate both the model and the user interface using appropriate approaches and reflect limitations.	LO1, LO5, LO8	RQ1, RQ2, RQ3

Table 1.11.1: Research objective

1.12 Chapter Summary

In this chapter, the necessary evidence of the problem and domain, research gaps, research challenge, aim, novelty and contributions were addressed, and the primary research questions and objectives are mapped with learning objectives.

CHAPTER 2: LITERATURE REVIEW

2.1 Chapter overview

This chapter aims to analyze and review the most significant existing literature on software vulnerability detection to understand the rise and impacts of software vulnerabilities. Additionally, it aims to identify challenges in software vulnerability detection by reviewing the state-of-the-art detection approaches. The author also analyzes the limitations of existing approaches and how to evaluate and benchmark a software vulnerability detection model.

2.2 Concept map

During the literature review, a concept map was maintained in parallel to simplify the research process by visualizing an overview of the research. The concept map grew over time to keep up with the literature review, and the latest version has been attached in the Appendix A – Concept map for reference.

2.3 Software vulnerabilities

2.3.1 The rise of software vulnerabilities

The recent technological advancements in the world have resulted in a rapid increase in the amount of software being developed. However, these advancements have also led to hidden flaws in such software, creating vulnerabilities that have caused a massive peak in vulnerability reports around the world. Despite that the reports on vulnerabilities have reached all-time high in recent years and continues to trend upwards, with no signs of decreasing (Skybox Security, 2022; Zero Day Initiative, 2023).

2.3.2 Impacts of software vulnerabilities

Software vulnerabilities can have a drastic impact on organizations and individuals, including financial losses, denial of service, reputational damage, data loss, and legal issues. In some cases, a single vulnerability can cripple an entire business.

For instance, LastPass, a password manager experienced a data breach in August 2022. Over 25 million users' personal information was compromised, including their names, email addresses,

billing addresses, and encrypted passwords. A vulnerability in LastPass's development environment was responsible for the breach. LastPass has since patched the vulnerability and is investigating the breach with law enforcement. However, the damage had already been done, and LastPass has lost millions of users and continues to lose (Coker, 2023; Tomaschek, 2023). And also according to Aiyer et al., (2022) at the current rate of growth, the annual cost of cyberattacks will be around $10.5 trillion by 2025, and organizations worldwide spent around $150 billion on cybersecurity in 2021, with that figure expected to rise further.

Having a reliable software vulnerability detection methodology is critical in order to prevent massive disasters in the future. Therefore, it is necessary to study the current state-of-the-art detection techniques and evaluate them to identify their strengths and weaknesses.

2.4 State of art software vulnerability detection

The current software vulnerability detection methodologies can be primarily classified into two major categories: code analysis-based detection and data-driven detection. (Lin et al., 2020).

2.4.1 Code analysis-based detection

Code analysis-based detection is a fundamental technique that can be further categorized into several sub-techniques, including static analysis, dynamic analysis, hybrid analysis, fuzz testing, and traditional analysis. Static analysis examines the source code of software without running it and can be used to detect vulnerabilities that are not exposed when the application is running. Dynamic analysis examines the behavior of software while it is running and can be useful to detect vulnerabilities that are exposed when it is running. Hybrid analysis combines both static and dynamic analysis. Fuzz testing puts random and unexpected inputs to the software to see how it reacts, which can be used to detect vulnerabilities that can be caused by unexpected inputs. Traditional analysis involves manually reviewing the code for potential vulnerabilities (Cowan et al., 1998; Hanif et al., 2021).

Each of these code analysis techniques has its own limitations, such as a high false positive rate, low code coverage, time consumption, the need for a level of expertise, and in most cases, they become inefficient when considering the massive recent spike in software vulnerability

reports. Therefore, the necessity for new, more sophisticated techniques has emerged. (Ghaffarian and Shahriari, 2018; Lin et al., 2020; Hanif et al., 2021).

2.4.2 Data driven detection

In addition to code analysis techniques, researchers are looking into data-driven approaches like data science and AI to improve software vulnerability detection. While data science-based methods are widely used in other fields, they are relatively new in the field of software security and have recently gained significant attention due to their effectiveness (Russell et al., 2018). Large datasets are used to train machine learning models that can identify patterns and anomalies indicative of software vulnerabilities. Unlike traditional code analysis techniques, data-driven approaches have the potential to provide higher accuracy, cover a larger portion of the codebase, and reduce false positives (Ghaffarian and Shahriari, 2018).

Although this field has its own challenges, data scarcity is one of the most obvious ones, and it is a major controlling factor (Lin et al., 2020). Another significant challenge is that even after using a data-driven approach, it can be difficult to interpret the final output, which in most cases is not easily understandable, like labels in arrays or lists (Ghaffarian and Shahriari, 2018). Additionally, there are other challenges, such as the ever-growing nature of this problem, which requires continuous involvement and improvement. There is also a high risk of misclassification, which can defeat the whole purpose (Bilgin et al., 2020).

2.5 A deep dive into significant related works

Taxonomy of software vulnerabilities detection and machine learning approaches (Hanif et al., 2021): This paper discusses the growing problem of software vulnerabilities, its impacts, and the existing approaches in use. The significance of this paper lies in analyzing all current approaches, comparing and contrasting their pros and cons. The paper shows how effective machine learning approaches are compared to code analysis techniques. It concludes that the future of software vulnerability detection relies on machine learning approaches and promotes more research to be done in this area. The paper also addresses challenges such as the scarcity of labeled datasets and difficulty in interpreting the final outputs. The most significant contribution of this paper is that it clearly paves the path for researchers to start research in software vulnerability detection.

VulDeePecker (Li et al., 2018): This was the first deep learning-based approach published, which involved tokenizing the source code and applying a deep neural network for feature extraction. Feature extraction was done at the slice level rather than the function level, and the authors primarily used BLSTM, a variant of RNN, for this purpose. Their primary finding was the ability to perform slice-level feature extraction, which is more precise than function-level extraction. However, while there was no significant improvement in the final results, the lack of explanation for the higher number of false positives and false negatives raises questions about the credibility of the results. Additionally, the dataset they used consisted of C and C++ functions extracted from various GitHub repositories, with a primary focus on library/API level functions, and it only contained two types of vulnerabilities, which could be considered a limitation.

Automated Vulnerability Detection in Source Code Using Deep Representation Learning (Russell et al., 2018)**:** The significance of this research is that it is the first to come up with a function-level labeled dataset and has proven that deep learning approaches are promising and accurate compared to traditional machine learning approaches. Their detection process includes three major stages:

- Lexing: The process of lexing the source code into a sequence of tokens.
- Feature extraction: They used a deep neural network to extract features from the tokenized source code, using both CNN and RNN approaches. According to the results, CNN outperformed RNN in every aspect of the research.
- Classification: The features were classified using classifiers like Random Forest, but this process was only used in their ensemble learning approaches.

In the end, CNN models without any ensemble learning approaches outperformed every other approach, clearly demonstrating that deep learning approaches can be used in software vulnerability detection.

The dataset created for this research was made publicly available and consists of 1.2 million C and C++ functions from publicly available repositories on GitHub, labeled as vulnerable and non-vulnerable. This dataset has become a standard dataset for software vulnerability detection problems and has been further improved by contributions from the OSF community, continuing to improve to date.

	SATE IV	GitHub	Debian
Total	121,353	9,706,269	3,046,758
Passing curation	11,896	782,493	491,873
'Not vulnerable'	6,503 (55%)	730,160 (93%)	461,795 (94%)
'Vulnerable'	5,393 (45%)	52,333 (7%)	30,078 (6%)

Figure 2.5.1 - Overview of original dataset (Russell et al., 2018)

The imbalanced nature of the dataset and the use of a single output layer in the neural network can be considered as limitations of this research. The research only focused on detecting a single vulnerability at a time, and while it is evident that detecting multiple vulnerabilities would be a future research area, this was not explored in this study.

Vulnerability Prediction From Source Code Using Machine Learning (Bilgin et al., 2020): The significance of this work lies in the use of Abstract Syntax Tree (AST) to represent the source code as a graph, where each node represents a different function or variable. This approach introduces a new step after tokenizing the source code, which involves generating an AST from the tokenized source code, converting it to binary values and encoding it as a tuple. For feature extraction, the author uses Multi-Layer Perceptron (MLP). The results were not as impressive and accurate compared to other similar work. However, this new approach is noteworthy and can be used in other research areas like code completion, according to the author. Interestingly, this research addresses five pre-defined vulnerabilities which is a major contribution.

SySeVR (Li et al., 2022): This research proposes a framework for the software vulnerability detection problem by comparing multiple deep learning approaches. The author of this research also worked on VulDeePecker and primarily focused on addressing the issues found in that. To test various deep learning approaches, the author used their own labelled dataset extracted from NVD and SARD, two major sources that include various vulnerable pieces of code from different software. These code pieces are labelled as good, bad, and mixed. The research found that bidirectional RNN approaches produced good results in edge cases; however, their evaluation suggests that CNN shows consistent performance in most cases. The research suggests that the use of semantic and syntax-based approach can be used for code representation; however, until discovering a reliable algorithm, it is appropriate to use function-level approach. The research also

suggests that detecting multiple vulnerabilities would be the ideal solution for this problem, even though this framework is proposed for program-level vulnerabilities.

2.6 Technologies for software vulnerability detection

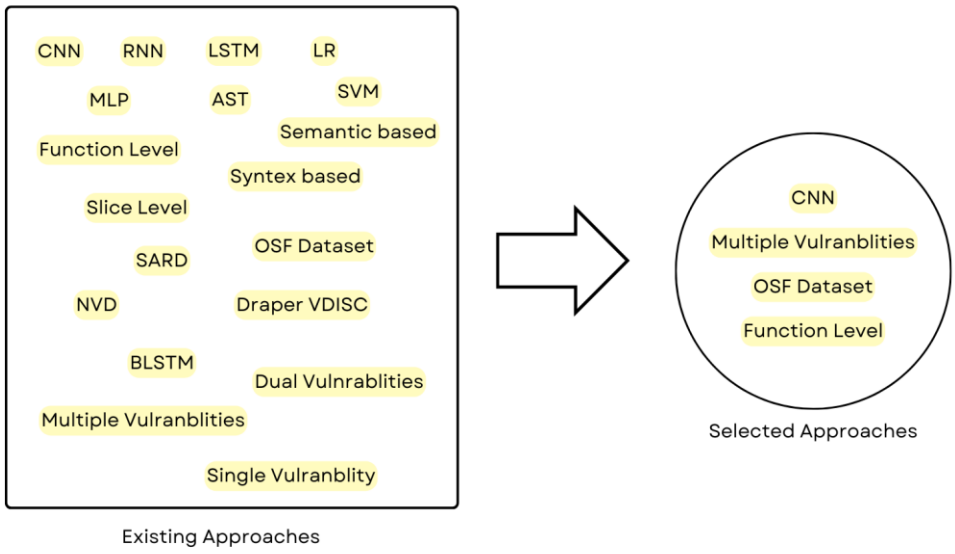


Figure 2.6.1 - Technologies for software vulnerability detection (Self-composed)

The figure above shows an overview of the technologies selected for each stage of the detection process, and all existing approaches were analyzed in the previous topic (2.5 It is ideal to review the selected approach here.

Tokenization: Tokenization is the process of creating a sequence of tokens out of the source code input. Different researchers have followed different approaches to handle source code tokenization, such as AST, slice-level tokenization, semantic and syntax-based representation, and function-level source code tokenization. However, it is clear that function-level source code tokenization is the best approach for deep learning-based detection techniques and performs well (Russell et al., 2018; Bilgin et al., 2020; Hanif et al., 2021; Li et al., 2022).

Feature extraction: Researchers have come up with various approaches to do feature extraction, such as traditional ML (e.g., LR, RF), RNN, bidirectional RNN, LSTM, BLSTM, and CNN. However, most researchers and their evaluations clearly show that CNN is the most reliable and best-performing architecture for deep learning-based detection of software vulnerabilities

(Hanif et al., 2021; Li et al., 2022). Some researchers even analyzed ensemble learning approaches by combining CNN with traditional ML approaches for classification, even though CNN without any ensemble learning combinations showed higher performance in comparison, which suggests that CNN itself would be the ideal choice for this problem. However, it is proposed to experiment with hyperparameters and multiple output layers to come up with a more sophisticated multiple vulnerability detection technique (Russell et al., 2018; Bilgin et al., 2020).

Dataset review: Due to the scarcity of labeled datasets for software vulnerability detection, only the OSF dataset (Draper VDISC Dataset), which was initially created by Russell et al., (2018) and further improved by the contributions of the OSF community, can be considered as a standard dataset for approaching this problem. This dataset contains C and C++ functions that are labeled as vulnerable and non-vulnerable. The properties of this dataset are discussed in depth in the previous section (2.5), as well as in the Dataset selection section. The unbalanced nature of the OSF dataset is an issue that many researchers lacked to address, which can cause biases in the final results, and this issue is aimed to be addressed. Some researchers used other datasets, such as SARD, and NVD, which were extracted from the same GitHub repositories, in addition to the OSF dataset (Li et al., 2018).

2.7 Evaluating software vulnerability detection.

Metric	Description
True positive rate (TP)	These are instances where the software vulnerability detection model has correctly detected a vulnerable sample. Higher the value better the model.
False positive rate (FP)	These are instances where the software vulnerability detection model has misclassified a non-vulnerable sample as vulnerable. Lower the value better the model.
Accuracy	Total instances of where the software vulnerability detection model correctly classified vulnerable and non-vulnerable samples respectively. Higher the value better the model.

Precision	The relevance of the model to predict vulnerable sample as vulnerable. High score means high relevance.
Recall	The rate of the model predictions to correctly predict vulnerable samples as vulnerable. Higher the value higher the relevance.
F1 score	The mathematically accepted average of both recall and precision. This will be an acceptable indication of how robust and reliable the software vulnerability detection model. Value 1 is the best and 0 is the worst.

Table 2.7.1 – Primary evaluation metrices

Previous works suggest that it is ideal to use the matrices mentioned above to evaluate a software vulnerability detection model, as they can clearly show the model's performance and reliability (Hanif et al., 2021; Singh, Grover and Kumar, 2022). Additionally, some researchers also suggest that false negative and true negative rates can also be useful to find out how the model classifies non-vulnerable cases, which can be an indication of the model's performance (Li et al., 2022).

2.8 Issues in benchmarking

Existing works highlight the issue in benchmarking due to the scarcity of data. The absence of a benchmarking dataset prevents the opportunity for benchmarking the proposed approaches (Ghaffarian and Shahriari, 2018; Hanif et al., 2021). However, some researchers suggest that benchmarking is possible only by reproducing previous work and comparing it with it. However, this method is entirely dependent on the reproducibility of previous work (Singh, Grover and Kumar, 2022).

2.9 Chapter summary

In this chapter, the author discusses the rising problem of software vulnerability detection and its impacts by analyzing various existing works. Additionally, the chapter covers current detection approaches along with their limitations. The author also reviews the technologies that are in use and how they can be evaluated. Finally, the chapter discusses the issues in evaluating a model without a benchmarking dataset.

CHAPTER 3: METHODOLOGY

3.1 Research methodology

The research methodologies that were picked in order to complete the project quickly and affordably are listed in the following table:

Philosophy	**Positivism** was chosen as the research philosophy because the study does not reflect any personal beliefs, and even though the author intended to collect qualitative data, all such data was intended to be converted into quantitative data in the end.
Approach	Since the purpose of the research is to test an established hypothesis, the **deductive** approach was chosen.
Strategy	Since research is intended to test a predetermined hypothesis, **experimental research** has been chosen and **archival research** methodology was also used because the research also examined a variety of previously published data and literature. Additionally, a **survey research** strategy has been used to collect data via questionnaires and interviews.
Choice	Since questionnaires and interviews are to be facilitated, In terms of questionnaires and structured interviews, the choice of method is **multi**.
Time horizon	Given the need to collect data at various points during the research, such as before and after implementation, **longitudinal** research was chosen as the time horizon for the study.
Data collection	Surveys, questionnaires, interviews, statistics, pie charts, pivot charts, organizational reports and records, annual forecasts, and legitimate news sources have been chosen to be used as data collection techniques and procedures.

Table 3.1.1: Resource requirements

3.2 Development methodology

As a software development model, incremental prototyping is chosen, in which a prototype is built, tested, and reworked until an acceptable prototype is achieved.

3.2.1 Requirement elicitation methodology

Surveying has been chosen as a method of gathering requirements. It has also been proposed to use structured questionnaires, and structured interviews to gather requirements. Virtual tools such as Google forms and Google meet will be used for these purposes. Experiments, both controlled and uncontrolled, will also be used to test the hypothesis. Besides that, literature reviews will be continued throughout the research period in order to stay current on the field and grasp current research requirements.

3.2.2 Design methodology

Throughout the project, agile design has been chosen as the design methodology. Because it is flexible enough to build, measure, learn, and repeat the process until a desirable design is achieved, and it also incorporates the Project management methodology.

3.2.3 Programming paradigm

The multi-paradigm approach was chosen since the project demands the use of multi-paradigm concepts, languages, and frameworks (such as Python, JavaScript, etc.).

3.2.4 Testing methodology

It is intended that prototype testing be carried out by presenting the prototype to targeted users and collecting their feedback. Additionally, the model is to be assessed using the proper matrices such as F1 score, Confusion matrix etc. The benchmarking will also be investigated by contrasting the suggested method with state-of-the-art approaches.

3.2.5 Solution methodology

It is necessary to follow end-to-end artificial intelligence research procedures as an artificial intelligence-based solution. There is a data collection process for developing a solution where the

synthetic datasets for known vulnerabilities are identified and data is entered into the data processing stage where the data cleaning and handling of missing data is done and also the feature engineering of data is completed after the model training where the algorithm is selected and evaluated with the selected data and finally the model is deployed.

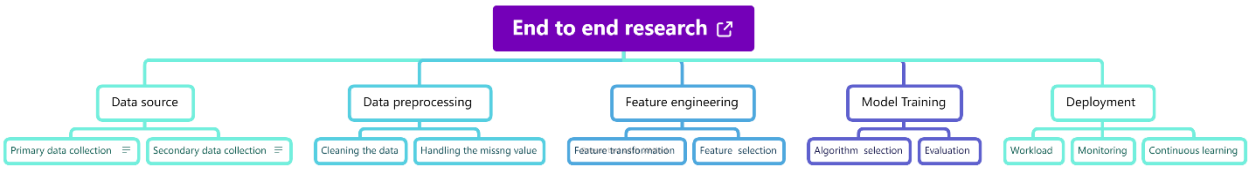


Figure 3.2.1: End-to-end research diagram (Self-composed)

3.3 Project management methodology

Kanban has been chosen as the project management methodology since the author is well-versed in Kanban and all major tools, including GitHub and Microsoft To-Do, have built-in Kanban board support to visualize and manage tasks.

3.3.1 Schedule

3.3.1.1 Gantt chart

The Gantt chart has been moved to Appendix B – Ganntchart.

3.3.1.2 Deliverables

Deliverable	**Date**
Project proposal	November 3, 2022
Literature review	October 27, 2022
Software requirements specification	November 24, 2022
Project specifications design and prototype	February 2, 2023
Design document	January 23, 2023
Prototype	January 17, 2023
Evaluation	March 23, 2023
Final report	May 10, 2023

Table 3.3.1 - Deliverables

3.3.2 Resource requirements

The following lists the hardware, software, data, and skill requirements that were necessary to finish this project.

3.3.2.1 Software requirements

- **Operating system (Windows/Linux/macOS) –** Linux is the best platform for building, training, and testing the model, while Windows is best for developing the user interface and documentation.

- **Python/R –** Python will be chosen over R because of its low learning curve and wide availability of machine learning libraries.
- **Google colab/ Jupiter –** Google colab will be preferred over Jupiter due to the availability of virtual machines, which will significantly improve execution time and resource usage.
- **TensorFlow/ Keras /Conda –** Keras and TensorFlow were chosen for their compatibility with Google collab and speed; flexibility was also taken into account.
- **JavaScript/Kotlin/PHP –** Because of its versatility, JavaScript was chosen to implement the web interface.
- **VS code/Fleet/Sublime/IntelliJ/PyCharm –** VS Code and Fleet were chosen over other code editors and IDEs because of their low weight and high level of compatibility.
- **Zotero/Research rabbit /Mendeley –** Due to its open-source nature, compatibility with other research tools like Research Rabbit, and support for an enormous number of add-ons, Zotero has been chosen over Mendeley.
- **GitHub/GitLab/Bitbucket –** Due to its extensive selection of integrated actions and user-friendliness, GitHub has been chosen.
- **Google Drive/One Drive/Dropbox –** Google drive was chosen due to its high compatibility and reliability.
- **Office 365/Google workspace/Canva/SparkPost –** Because of their ease of use and feature richness, Office 365 and Canva were chosen for documentation and the creation of diagrams and presentation materials, respectively.

3.3.2.2 Hardware requirements

To complete high-resource-demanding tasks such as training, building, and testing models, as well as storing and managing data, the following hardware requirements must be met.

- Processor: at least Intel(R) Core (TM) i7
- RAM: at least 8 GB usable
- GPU: NVIDIA GeForce MX130 or above
- Diskspace: at least 50GB

3.3.2.3 Data requirements

- Code samples with synthetic or real-time know vulnerabilities. – From GitHub and previous works.
- Code samples without vulnerabilities – From Kaggle, GitHub and previous works.

3.3.2.4 Skill requirements

- Academic writing.
- Ability to build the required artificial intelligence model
- Ability creates user interface and integrates the model.

3.3.3 High-risk items

Identified risks have been ordered and listed below based on the risk exposure:

Risk	**Probability**	**Magnitude**	**Mitigation plan**
Losing access to files, documentation, and source code because of hardware problems or other problems.	5	5	Maintaining a GitHub repository and fetching on a regular basis and integrating Google drive for desktop to automate file backups.
Unable to complete the research due to sickness.	4	5	Maintain a healthy lifestyle and prioritize core functions to achieve a minimum viable product sooner.
Unable to complete the research due to lack of knowledge.	4	5	Consult with experts to obtain learning resources and to identify resources through LR, as well as attend all academic help sessions.

Unable to complete the research within the allocated research period.	5	4	Prioritize core functionalities and maintain a progressive Kanban board.
Unpublished research with a similar approach got published and pop out of nowhere	4	5	Continuous literature review and ensure the novelty of the approach.

Table 3.3.2: Risk management

CHAPTER 4: SRS

4.1 Chapter Overview

This chapter is based on identifying potential stakeholders through the use of rich picture and onion diagrams in order to gather requirements using appropriate instruments and identify their use case in order to develop functional and non-functional prototype requirements.

4.2 Rich Picture Diagram

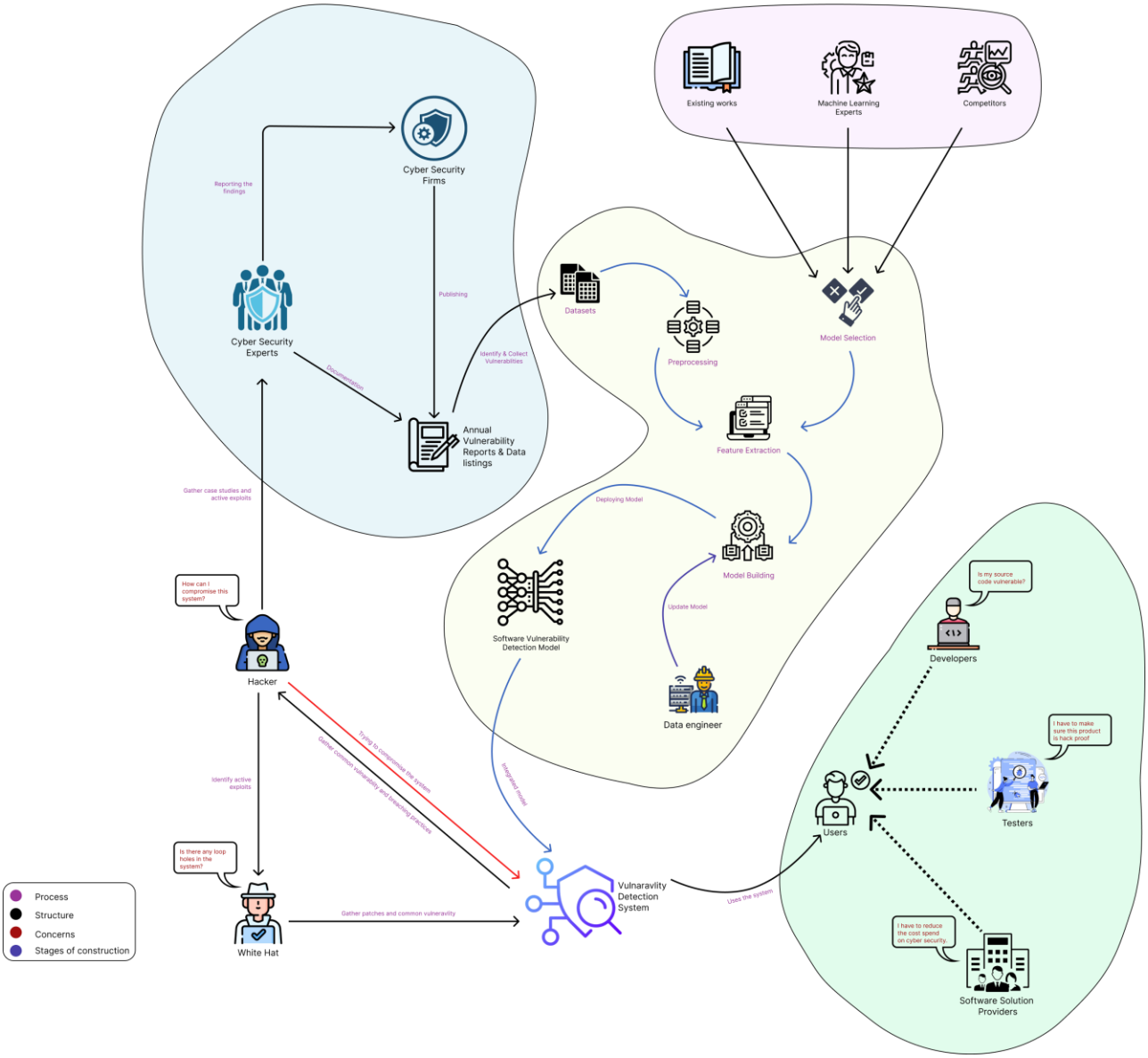


Figure 4.2.1 - Rich Picture (Self compose)

Figure 4.2.1 is the rich picture of the project which is an overview of the project including structures, processes and concerns related to the project and this will be used to identify stakeholders of this project.

4.3 Stakeholder Analysis

4.3.1 Stakeholder onion model

An overview of the stockholders identified from the rich picture has been displayed with an onion diagram in Figure 4.3.1.

Figure 4.3.1 - Stakeholder onion (Self-composed)

4.3.2 Stakeholder viewpoints

Stakeholders	Roles	Description
Product owner	System owner	Owner of the system and gather all requirements from other stakeholders and provide it to relevant stakeholders
Application Developer	Maintenance	Develop and maintain the system
Solution providers	Functional beneficia/ Normal operator	Use the system and can act as an investor and also can be a normal operator when they use the system to check for vulnerabilities.
Testers	Quality regulator advisors / Normal operator	Evaluating the quality and functionality of a product or system can also be a user when they use the system to detect vulnerabilities for their system and also can be a normal operator when they use the system to check for vulnerabilities prior to deployment.
Cyber security firms	Technical experts	Provide insights on common vulnerabilities and the scope of the project and introduce state of art tools.
Cyber security experts		Provide expertise, opinions, and continuous feedback to improve the system regarding the problem domain.
Machine learning experts		Provide expertise, opinions, and continuous feedback to improve the system regarding the research domain.

Ethical hackers		Provide insights on state of art techniques used to breach a system through vulnerabilities and which vulnerability to give priority.
Hacker	Negative stakeholder	Those who try to compromise this or any other system. By learning past case studies the author can gather requirements.
Competitor		Those who build competitive systems that try to outperform the system.
Researchers	Functional beneficia	Can use the system to strengthen the defenses of their systems and also provide literature insights.

Table 4.3.1 - Stakeholder viewpoints

4.4 Selection of Requirement Elicitation Methodologies

Based on the identified stakeholders to gather requirements interviews, literature review and brainstorming has been selected as gathering instruments. Reason and relevant stakeholders have been provided below.

Method 1: Literature Review
Literature review is a methodology used in software vulnerability research to gather information about existing vulnerabilities and solutions. It involves searching and analyzing research papers, articles and publications related to the software system being evaluated, in order to identify potential vulnerabilities and understand how to detect them. The goal is to gain a deep understanding of the problem and existing solutions and to identify any gaps or limitations that need to be addressed in the detection process. It is an important step in the vulnerability detection process and helps to ensure that the methods used are well-informed and effective.
Method 2: Interviews

Interviewing experts is a methodology used in software vulnerability detection projects, where experts in cybersecurity, machine learning and software testing are interviewed to gather information about common vulnerabilities and best practices for detection. The goal is to gather insights and information to help design and validate the vulnerability detection system. It's an important step to make sure that the system is well-informed and effective.
Method 3: Brainstorming
Brainstorming with fellow researchers and software developers is a methodology where a group of researchers and developers come together to share ideas and generate solutions for identifying and mitigating vulnerabilities in software systems. The goal is to generate a wide range of ideas and solutions and identify any potential vulnerabilities that may have been overlooked

Table 4.4.1 – Requirements elicitation methodologies

4.5 Discussion of Findings

4.5.1 Interviews

Coding	Theme	Finding
Detection, Tools	Tools/Methods for Detection	Respondents currently use traditional methods such as manual inspection, dynamic analysis and commercial tools
Phases	Ideal Phase for Detection	The majority of respondents believe the ideal phase for detecting vulnerabilities is prior to deployment
Common, Severe	Common Vulnerabilities	The most common vulnerabilities mentioned by respondents include SQL injection, cross-site scripting (XSS) and cross-site request forgery (CSRF)

Impactful	Threatening Vulnerabilities	The most threatening vulnerabilities mentioned by respondents include those that allow attackers to take control of systems and steal sensitive data
Finance	Budget for Cybersecurity	Respondents reported varying budget allocations for cybersecurity in their organizations, with some having a dedicated budget and others relying on a general IT budget
Techniques	Ideal Algorithms/Techniques	Respondents believe that AI and machine learning techniques such as neural networks, deep learning and reinforcement learning have the potential for ideal software vulnerability detection, but also believe that a combination of techniques is required for effective detection.

Table 4.5.1 - Findings from interviews

4.5.2 Literature review

Citation	Findings
(Hanif et al., 2021)	Software vulnerabilities can be a disastrous problem which can totally collapse businesses and the recent rise in vulnerabilities is skyrocketing in the industry.
(Sarker et al., 2020)	Using multi-layer machine learning can be used to detect software vulnerabilities and it is necessary to build a data-driven intelligence capable of making smart decisions for cyber security problems.

(Hu, 2019)	Deep learning can be used to identify software flaws. BLSTM may be ideal for detecting pointer vulnerabilities. It can be difficult to identify vulnerabilities in code.
(Singh, Grover and Kumar, 2022)	NLP can be used to detect known vulnerabilities. Detecting known vulnerabilities is currently a time-consuming process. When it comes to detecting vulnerabilities, BERT is very accurate.
(Li et al., 2022)	Neural networks can be used to detect software vulnerabilities in place of code similarity-based methods, which have high false-negative rates.

Table 4.5.2 - Findings from LR

4.5.3 Brainstorming

Criteria	**Findings**
Accuracy	AI algorithms should be able to accurately detect vulnerabilities in software with a high degree of precision and changing parameters and learning curves can improve the accuracy.
Speed	The software should be able to scan and detect vulnerabilities quickly, to minimize the window of opportunity for attackers. And by reducing the learning rate we can increase the speed, but it can come with the price of lowering accuracy so, it’s vital to find the balance.
Expectations	The software should be easy to use, even for those with limited technical knowledge and it should be reliable.
Features	The system should be able to do what it promises, and more language support is preferred.

Table 4.5.3 - Findings from Brainstorming

4.6 Summary of findings

Findings	LR	Interview	Brainstorming
Software vulnerabilities are a critical problem and existing solutions need a lot of improvement.	☑	☑	☑
Vulnerabilities should be priorities and the most important vulnerabilities to address are severe vulnerabilities.		☑	
AI techniques like machine learning, deep learning and neural networks can be used to detect software vulnerabilities.	☑	☑	☑
Severe vulnerabilities will be specific for technologies (OS, Frameworks etc.) and programming languages and they can be identified by going through annual reports like CWE reports.		☑	☑
Detecting software vulnerabilities prior to deployment would be the ideal phase for detecting vulnerabilities with minimum impact.	☑	☑	☑
True positive, False positive, recall, precision can be used to evaluate the model's reliability.	☑		☑

Table 4.6.1 - Summary of findings

4.7 Context Diagram

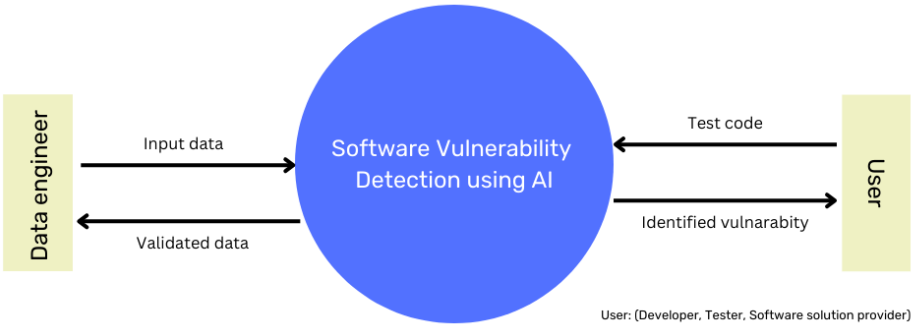


Figure 4.7.1 - Context diagram (Self-composed)

To simply understand the requirements a context diagram (Figure 4.7.1) show the interactions and relationships of a system with its external environment. It defines the scope and boundaries of the system and serves as a basis for further analysis. It can also be considered as a level 0 data flow diagram.

4.8 Use case diagram

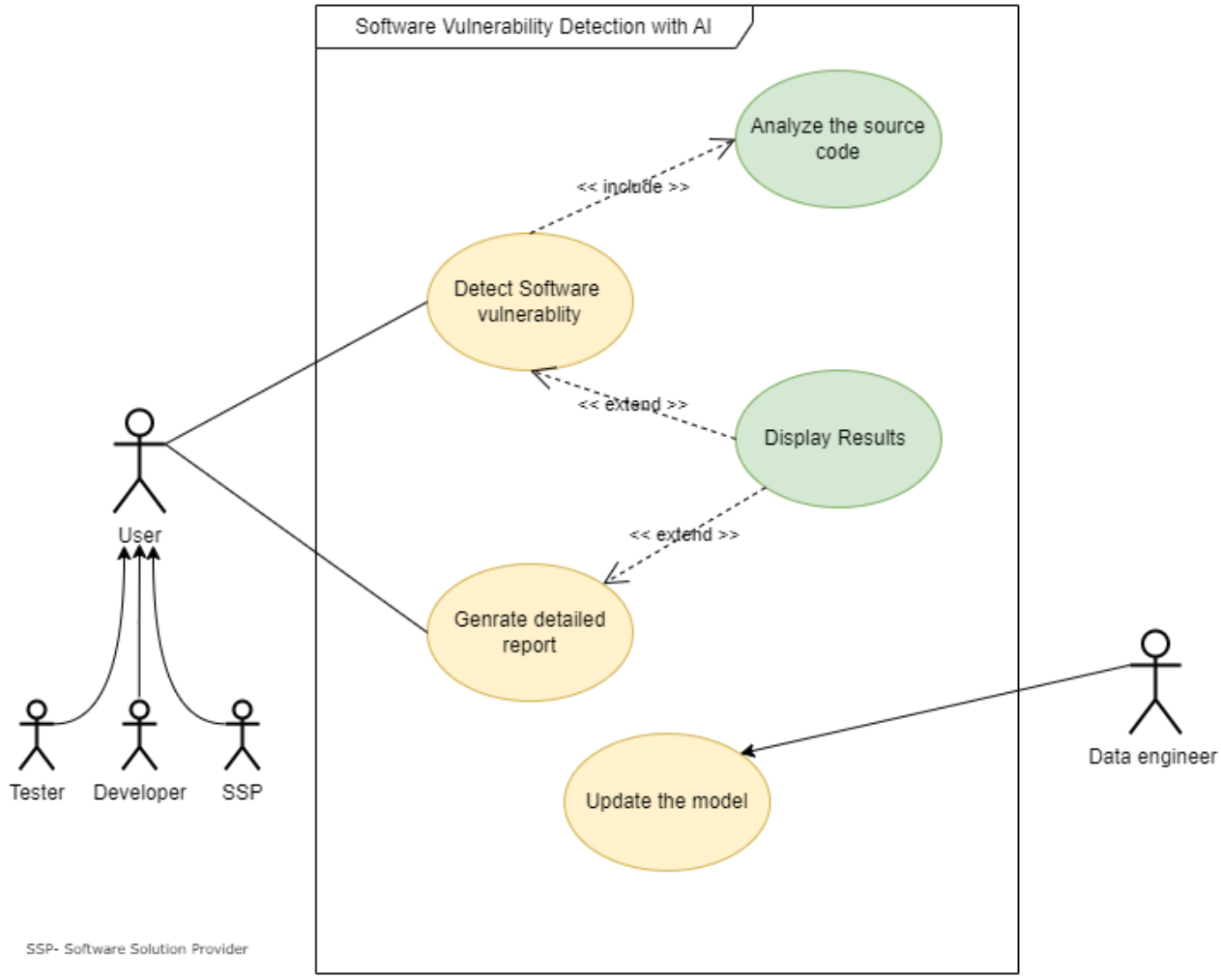


Figure 4.8.1 - Use case diagram (Self-composed)

Figure 4.8.1 is the use case diagram for the system which displays the primary use cases of the system with actors, use cases and relevant annotations.

4.9 Use case descriptions

Use case	Detect software vulnerability
Id	UC01

Description	Detect software vulnerabilities from provided source code.
Primary actor	User
Secondary actor	none
Precondition	The source code should be in supported language and within the expected character limit.
Postcondition	Success end: The user will be presented with results and potential vulnerabilities. Failure end: The user will encounter an error message.
Main flow	Open the application. Upload source code. Wait for the processing. View results.
Alternative flow	None
Exceptional flow	Language not supported. Exceeded character limit. No vulnerability was found. Encounter error.
Extended use case	Display results.
Included use case	Analyze the source code.

Table 4.9.1 - Use case description for UC01

4.10 Requirements

In order to prioritize gathered requirements MoSCoW technique has been used.

Priority Level	Description
Must have (M)	These are the core requirement and mandatory for the system.
Should have (S)	These are important requirements even though it's not critical to have them in a prototype,
Could have (C)	These are desirable to have requirements where it can be optionally implemented.
Will not have (W)	These requirements are simply considered out of scope.

Table 4.10.1 - MoSCoW level of priority

4.10.1 Functional requirements

FR ID	Requirements	Priority Level
FR 1	The system should perform a deep learning-based vulnerability analysis on the source code and identify any potential vulnerability	M
FR 2	The system should be able to detect multiple vulnerabilities	M
FR 3	The system should allow the user to paste the source code in a commonly used programming language	M
FR 4	The system should display primary vulnerability type found in the code in a meaningful way	S

FR 5	The system should also display secondary vulnerabilities found in a meaningful way.	C
FR 6	The developer must be able to update/select model without disrupting the system	C
FR 7	The system should include a confidence level and brief description of the vulnerability.	C
FR 8	The system should allow the user to view a more detailed description of a specific vulnerability.	C
FR 9	The system should allow the user to download a report of the analysis results in a common file format (e.g., PDF, HTML, etc.).	C
FR 10	The system should handle errors gracefully and display appropriate error messages in case of any problems (e.g., input failure, processing error, etc.).	C
FR 11	The system should have an authentication mechanism.	W

Table 4.10.2 - Functional requirements

4.10.2 Non-Functional requirements

NFR ID	Description	Requirement	Priority Level
NFR 1	The system should be accurate when detecting software vulnerabilities.	Accurate	M
NFR 2	The system should be reliable with lesser false positives and false negatives	Reliable	M

NFR 3	The system should have a user-friendly interface that is easy to use and navigate.	User friendly	S
NFR 4	The system should be scalable and capable of handling increasing volumes of source code as the number of users grows.	Scalability	C
NFR 5	The system should have a fast response time and perform the analysis quickly, even for large source code function.	Performance	C
NFR 6	The system should be compatible with a range of commonly used programming languages and file formats.	Compatibility	C
NFR 7	The system should be easy to maintain and upgrade to accommodate new features and improvements.	Maintenance	C
NFR 8	The system should be accessible to a wide range of users, including those with disabilities.	Accessibility	W

Table 4.10.3 - Nonfunctional requirements

4.11 Chapter summary

This chapter derives how the rich picture of the project evolves and from that stakeholders were identified and analyzed. Further to gather requirements the relevant tools were selected, and the findings were discussed. And also, the context diagram and use case diagrams were used to further elaborate the system.

CHAPTER 5: SLEP ISSUES

5.1 Chapter overview

This chapter discusses the social, legal, ethical, and professional concerns surrounding the research, as well as the mitigations implemented to address these issues. The University of Westminster Code of Practice Governing the Ethical Conduct of Research 2020-21 and the BCS Code of Conduct were referred to in order to learn the standards and practices necessary to ensure compliance with social, legal, ethical and professional guidelines.

5.2 SLEP issues and mitigation

5.3 Social issues

Although the end results of the software are reliable, there are instances where false positive and false negative classifications can occur. This may lead to inaccurate interpretation and can cause financial losses, as well as damage to the software's reputation and reliability, which defeats the purpose of the research. To avoid these issues, proper testing with real-world data and ideal validation of the software is necessary before commercial use. Furthermore, a confidence level has been implemented in the results (Figure 5.3.1) to narrow down the decision-making process of the user, which will serve as an effective mitigation for this issue.

Figure 5.3.1 - Sample implementation of confidence level

5.4 Legal issues

Legal responsibility has been considered throughout the research, particularly when conducting interviews. The consent of the participants was obtained, and all personally identifiable information was removed from the interpretation of such participation. The required dataset was

obtained from the Open Science Framework (OSF), which does not contain any personally identifiable information. To ensure compliance with legal requirements, the CC BY 4.0 license was obtained through a legitimate email from the author. Additionally, all software, tools, and resources used for this research are fully licensed through either personal or institutional privileges of the author.

5.5 Ethical issues

According to the Westminster Code of Practice, this research falls under Class 1, which pertains to research with no or minimal ethical implications, where risks will not exceed those experienced in normal day-to-day life. This type of research is considered to have no or minimal ethical implications and does not normally require ethical approval by a Research Ethics Committee (University of Westminster Code of Practice Governing the Ethical Conduct of Research, 2021).

Nonetheless, throughout the research, ethical clearance and practices were followed to maintain the code of conduct. Informed consent was obtained from all interview participants, and their names were used with their permission. No other unnecessary personally identifiable information was gathered. Furthermore, all referenced works and resources were properly cited and given credit.

5.6 Professional issues

The professional integrity of this research was ensured by adhering to the BCS Code of Conduct and following best practices. The author also remained open to reviews and constructive feedback from alternative viewpoints to improve the project. Staying informed on parallel technological advancements further refined the research project and demonstrated the author's professional competence. The research was conducted with the university's best interests in mind, and the author takes full responsibility for all their actions. To ensure data safety and confidentiality, all research-related resources were stored in strong biometric-protected devices, with regular backups kept in the university-provided Google Cloud. Additionally, implementations were stored in GitHub as a private project to maintain proper version control.

5.7 Chapter summary

This chapter addressed the crucial components of the social, legal, ethical, and professional concerns of this report. It also discussed the mitigation approaches taken to address these concerns.

CHAPTER 6: DESIGN

6.1 Chapter overview

This chapter is focused on the design of the system by defining design goals and from that getting the high-level and low-level architecture of the systems. This chapter also includes the author's design paradigm selection and relevant diagrams for that as well as the expected UI of the system.

6.2 Design goals

Design Goal	Description
Accuracy	The system should be able to accurately detect vulnerabilities in the software.
Coverage	The system should be able to detect a wide range of vulnerabilities, including both known and unknown vulnerabilities.
False positives	The system should minimize false positives, as they can be time-consuming to investigate and can decrease the credibility of the system.
False negatives	The system should minimize false negatives, as they can allow vulnerabilities to go undetected and increase the risk of a security breach.
Ease of use	The system should be easy to use and understand so that developers can easily incorporate it into their workflow.

Table 6.2.1 - Design goals

6.3 High-level design

6.3.1 Architecture diagram

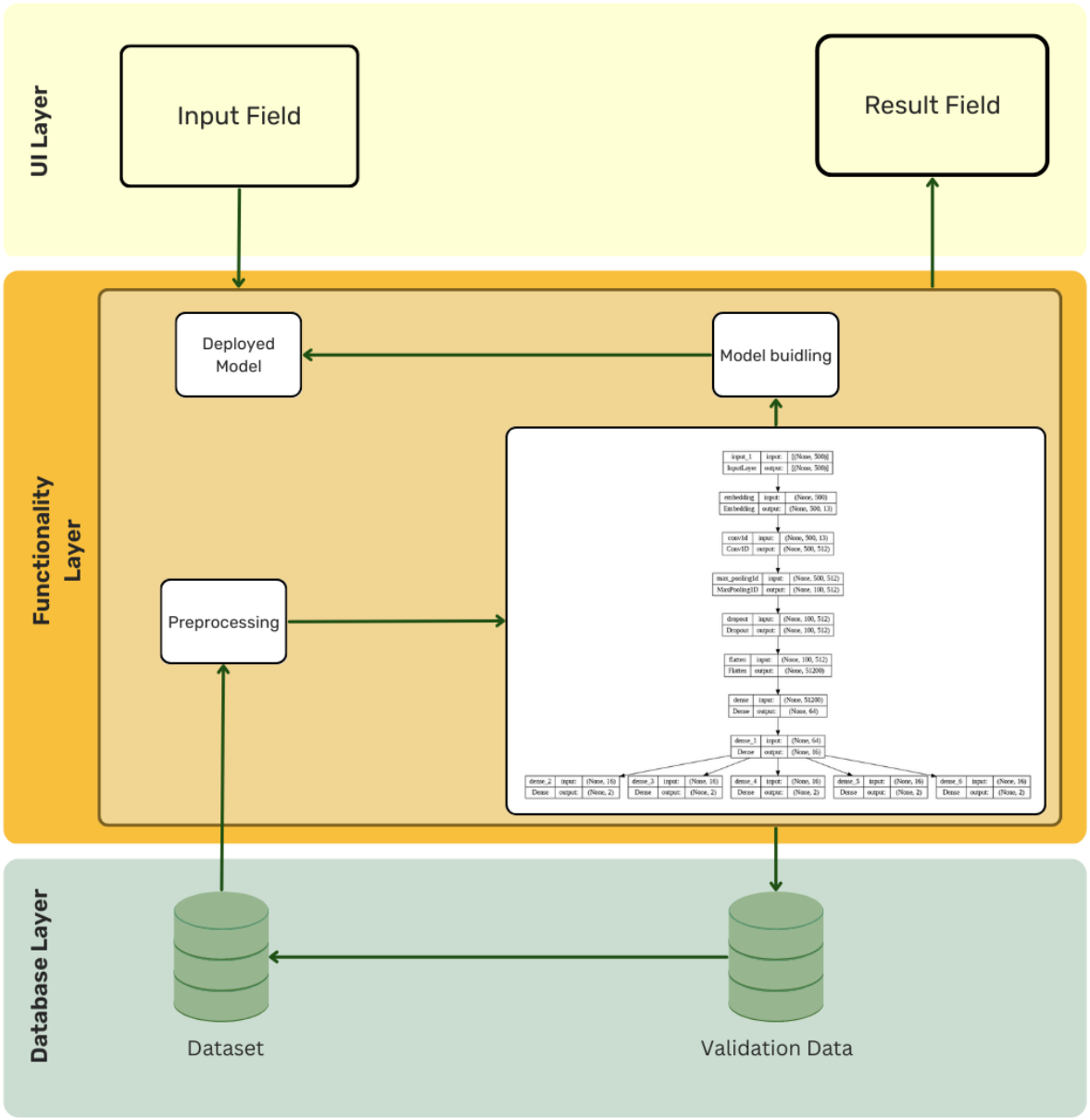


Figure 6.3.1 - Layered architecture(Self-composed)

6.3.2 Discussion of the layers

The layered diagram for the vulnerability detection system is comprised of three main layers: the UI layer, the functionality layer, and the database layer.

6.3.2.1 UI layer

The UI layer is the interface with which users interact. It has an input field for source code files and a result field for displaying the results of the analysis. The functionality layer is linked to the input field, which processes the source code and performs the vulnerability analysis.

6.3.2.2 Functionality layer

The functionality layer is where the core processing and analysis occur. This layer contains the model, which is responsible for preprocessing, feature selection, model building, and deployment. The model is connected to the result field in the UI layer to display the analysis results. The vulnerability detection process happens in this layer. Further explanation of the CNN layers will have been discussed in Implementation.

6.3.2.3 Database layer

The database layer contains the dataset and validation dataset. The dataset is used to train the model and the validation dataset is used to validate the model. The dataset is connected to the model in the functionality layer to provide the training data.

Each layer serves a specific purpose in the overall operation of the system, and they collaborate to provide an efficient and effective solution for AI-based software vulnerability detection. The separation of these layers into distinct components also makes future maintenance and upgrades easier.

6.4 Low-level design

6.4.1 Choice of design paradigm

SSADM (Structured Systems Analysis and Design Method) has been chosen to design the software vulnerability detection AI because of its structured and systematic approach, which can help ensure that all aspects of the system are thoroughly considered and that the final design meets

the requirements of the other hand OOAD (Object-Oriented Analysis and Design) is an object and class-based paradigm which is not appropriate for a growing system like this.

6.4.2 Data flow diagrams

A Level 1 Data Flow Diagram (DFD) is a graphical representation of a system's data flow. It divides the system into smaller, more manageable components and illustrates the relationships between inputs, processes, outputs, and storage as shown below.

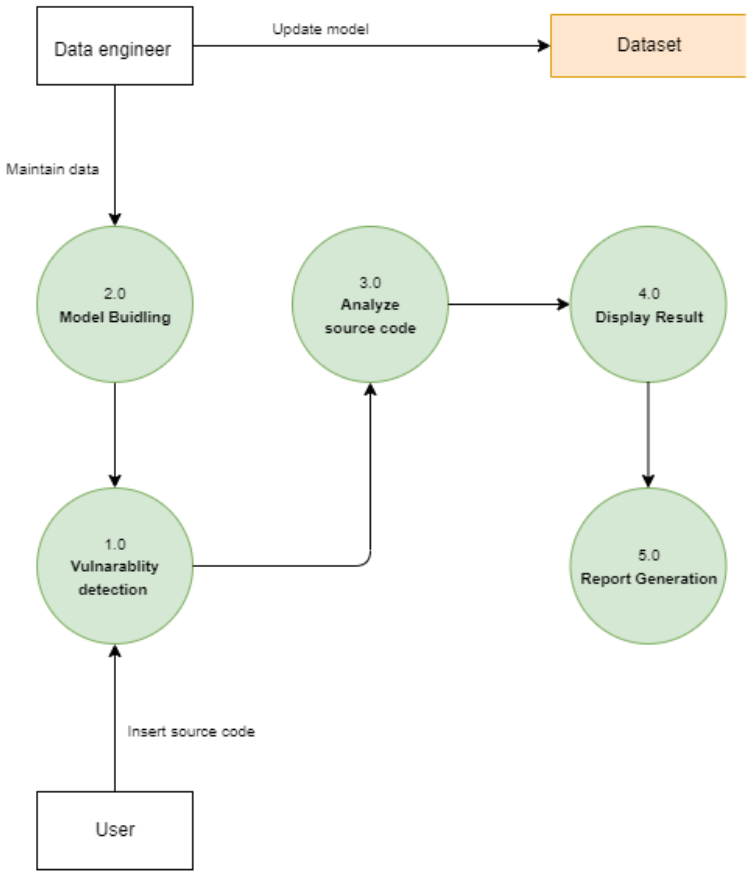


Figure 6.4.1 - Level 1 DFD (Self-composed)

A Level 2 Data Flow Diagram (DFD) is a more detailed representation of a system than a Level 1 DFD. It provides a deeper understanding of a system's processes by breaking them down into smaller, sub-processes and displaying the data flow between these sub-processes. Level 2 DFDs provide a comprehensive view of a system, allowing potential problems and areas for

improvement to be identified more easily. The following is the DFD for process 2.0 which is the main process of this system.

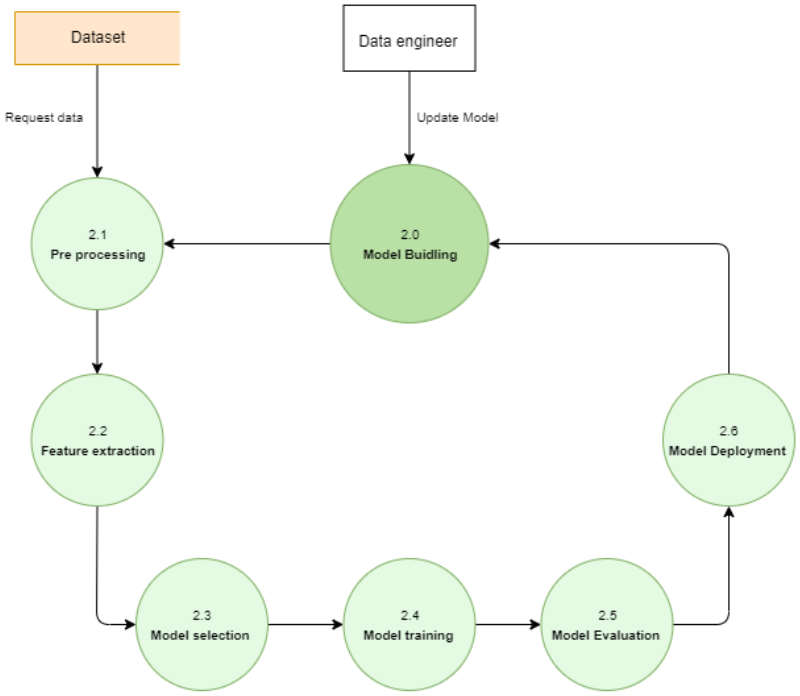


Figure 6.4.2 - Level 2 DFD (Self-composed)

6.5 Design diagrams

6.5.1 Component diagram

The following is a component diagram that shows the relationship between components in a system. It provides a high-level representation of how individual components work together to achieve a common goal.

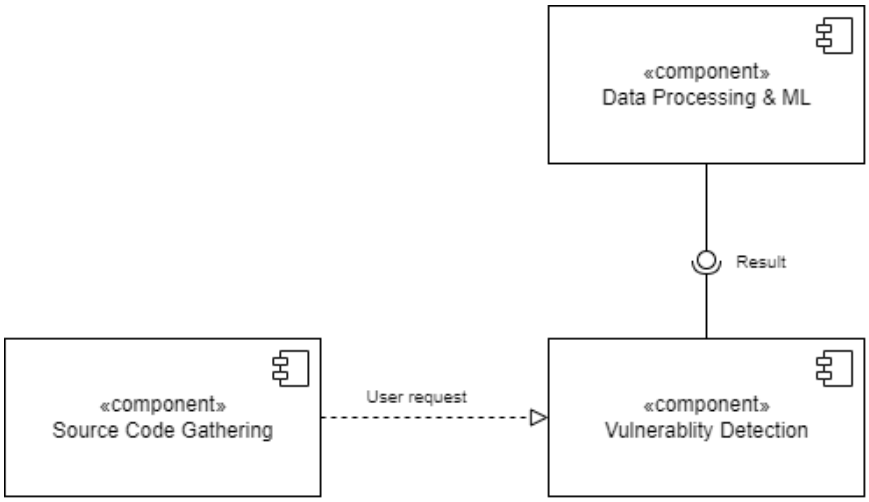


Figure 6.5.1 - Component diagram (Self-composed)

6.5.2 System process flowchart

The following is an activity diagram that shows the flow of activities in a system it models the steps taken to achieve a goal and it can be used to understand and improve the flow of activities in a system.

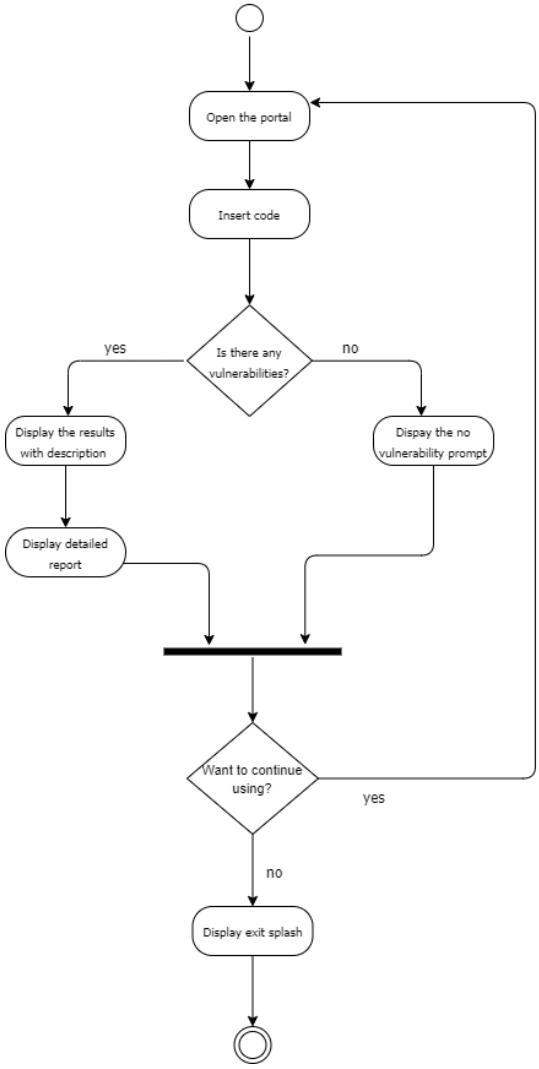


Figure 6.5.2 - Activity diagram (Self-composed)

6.5.3 User interface design

The system UI is intended have a place to have a main page with a place to upload source code, a review page to look up the upload process and a result page where the final result will be displayed along with the button to generate a detailed report as shown in the below wireframes.

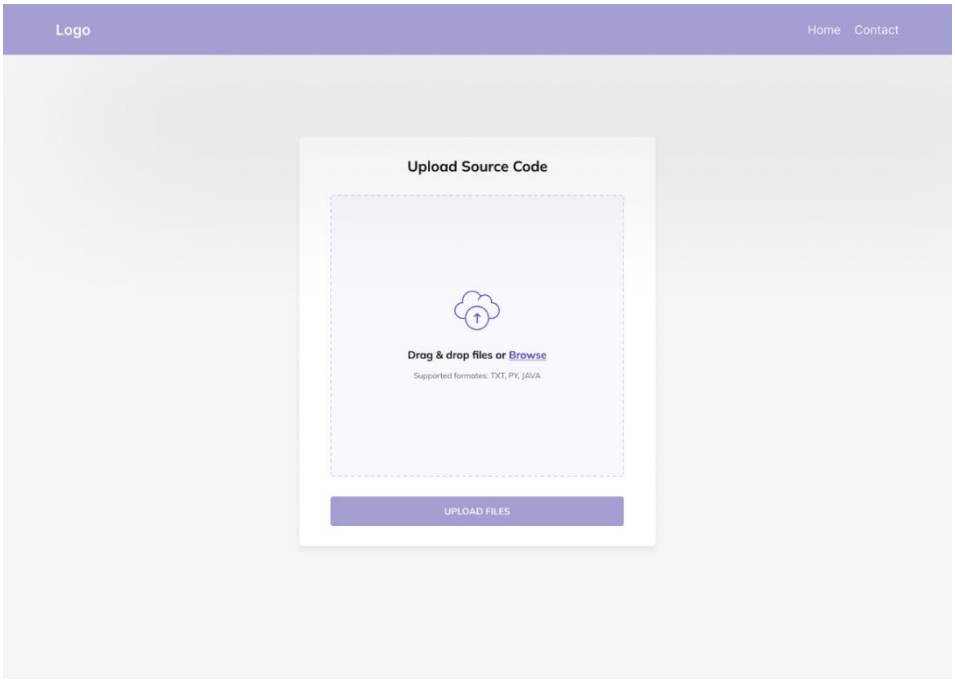


Figure 6.5.3 – Prototype Main Page UI (Self-composted)

And the prototype flow of the intended system can be viewed through this link as well to better understand the system. This protype UI has been changed based on the expert view and final design was implemented

6.6 Chapter summary

In this chapter, the design goals, architecture diagrams, design diagrams, and low-level prototypes for the vulnerability detection system were thoroughly documented. This documentation provides a clear understanding of the design and architecture of the system, including its various components and how they interact with each other and an overview of the user interface as well.

CHAPTER 7: IMPLEMENTATIONS

7.1 Chapter overview

T This chapter discusses implementation-related topics, including the technology stack, dataset selection, chosen development framework, programming languages, libraries, and IDEs used to develop the proposed solution.

7.2 Technology selection

7.2.1 Technology stack

The technology stack related to each layer has been pictured as an overview in the following diagram.

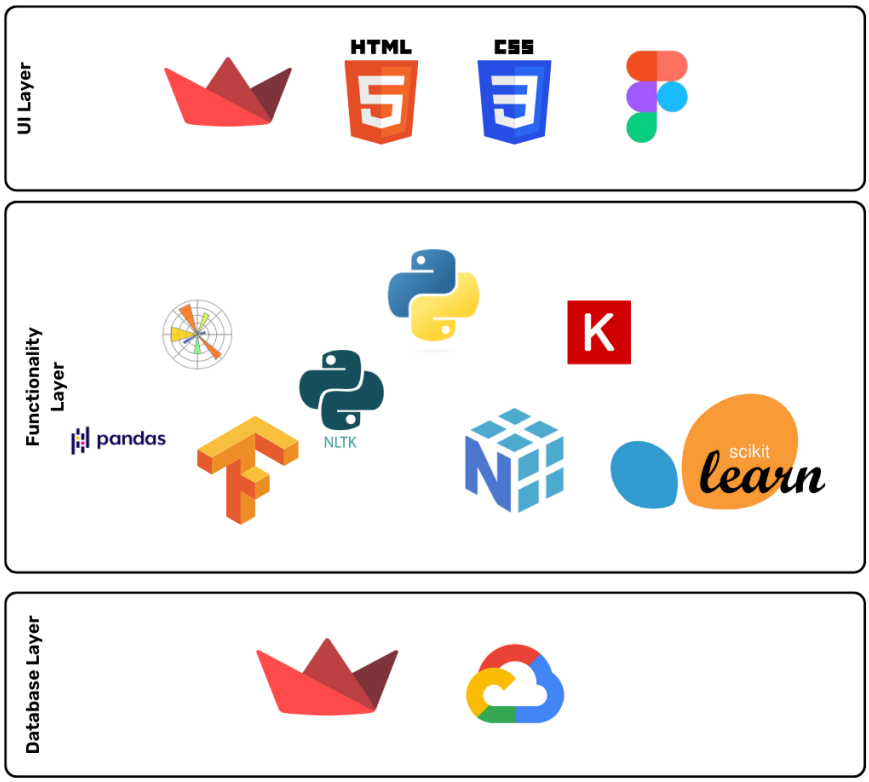


Figure 7.2.1 - Tech stack (Self-composed)

7.2.2 Dataset selection

The Draper VDISC dataset was chosen for this vulnerability detection system project. It contains 1.27 million functions from open-source software and has been labelled for potential vulnerabilities through static analysis. The data is divided into 80% for training, 10% for validation, and 10% for testing and is stored in 1 GB HDF5 files. Each function's source code is stored as a string and has five binary labels to indicate the presence of vulnerabilities such as CWE-120, CWE-119, CWE-469, CWE-476, and others. This dataset was created by a research group at Boston University for their research and open source to contribute to future research. (Russell et al., 2018)

7.2.3 Development framework

Framework	Justification
TensorFlow	TensorFlow is an open-source machine learning library that is widely used for building deep learning models. It is highly flexible and can be used for software vulnerability detection. TensorFlow is also supported by a large community, ensuring that any bugs or issues can be quickly resolved.
Flask	Flask is a lightweight web framework for Python that is ideal for developing small to medium-sized web applications. It is simple to use and highly customizable, making it an excellent choice for building the functionality layer of the vulnerability detection system. Flask also integrates well with other libraries such as TensorFlow and NLTK, making it a versatile and practical choice for the system.

Table 7.2.1 - Framework selection

7.2.4 Programming languages

Programming Language	Justification

Python	Python is widely used in the field of machine learning, data analysis and scientific computing. Its vast library support including TensorFlow, NLTK, Pandas and Skit learn makes it an ideal choice for developing a software vulnerability detection system.
JavaScript	JavaScript is widely used in web development, making it a suitable choice for creating the front end of the software vulnerability detection system using React. Its fast and dynamic nature makes it well-suited for creating interactive user interfaces and handling user inputs.

Table 7.2.2 - Programming language selection

7.2.5 Libraries

Library	Justification
Streamlit	Streamlit is a Python library that makes it easy to create beautiful, data-driven web apps in minutes. It's built on top of popular Python libraries like NumPy, Pandas, and Matplotlib.
Pandas	Pandas is a powerful and flexible library for data manipulation and analysis. It has a wide range of functionality, including data cleaning, data transformation, data aggregation and visualization. This makes it well suited for pre-processing and transforming the vulnerability detection dataset.
Skit learn	Skit learn is a machine learning library that provides a large collection of algorithms and tools for data analysis and modelling. It can be used for feature selection and building deep learning models for vulnerability detection.
NLTK	The Natural Language Toolkit (NLTK) is a library for NLP and text-processing tasks. It can be used for pre-processing and cleaning the source code data in the vulnerability detection dataset.

Numpy	Numpy is a library for scientific computing and data manipulation. It provides support for multi-dimensional arrays, matrices and mathematical functions, which are essential for deep learning models.
Matplotlib	Matplotlib is a data visualization library for creating plots, graphs, and charts. It can be used for visualizing the results of the vulnerability detection models.

Table 7.2.3 - Libraries selection

7.2.6 IDE

IDE	Justification
Visual Studio Code (VS Code)	VS Code is a popular, open-source IDE that offers a rich set of features, including support for multiple programming languages, debugging, and source control integration. It is lightweight and has a large community of developers who contribute to its extensions and plugins. This makes it a suitable choice for software development projects, including software vulnerability detection.
Google Colab	Google Colab is a free, cloud-based Jupyter notebook environment that supports Python programming. It allows you to run and share Jupyter notebooks without the need for any setup. It also provides access to powerful computing resources, such as GPUs and TPUs, which can be leveraged for machine learning tasks. This makes Google Colab a suitable choice for developing and testing deep learning models for software vulnerability detection.
GitHub Codespaces	GitHub Codespaces is a cloud-based development environment that integrates directly with GitHub. It provides a development environment that can be quickly set up, allowing you to focus on coding, testing, and

	debugging. Since it's a cloud-based platform it will be handy to use it along with other primary IDEs.

Table 7.2.4 - IDE selection

7.2.7 Summary of technology selection

Component	Tool
Programming languages	Python, JavaScript
Development framework	React, TensorFlow, Flask
Libraries	Pandas, Skit learn, NLTK, Numpy, Matplotlib, Pickle
IDE	VScode, Google collab, GitHub codespaces
Version control	Git, GitHub

Table 7.2.5 - Summary of technology selection

7.3 Implementation of core functionalities

The implementation consists of two parts. The first part is a notebook where each major stage of development is presented as a markdown table of contents. The second part is the app.py file, which defines the Streamlit app and connects it to the saved model. All necessary files have been uploaded to the GitHub repository, and the link to the repository is provided below. The primary implementation processes, including the layers of CNN, the module building process, and screenshots of app.py, are provided as snippets below.

```python
# Must use non-sequential model building to create branches in the output layer
inp_layer = tf.keras.layers.Input(shape=(INPUT_SIZE,))
mid_layers = tf.keras.layers.Embedding(input_dim = WORDS_SIZE,
                                       output_dim = 13,
                                       weights=[random_weights],
                                       input_length = INPUT_SIZE)(inp_layer)
mid_layers = tf.keras.layers.Convolution1D(filters=512, kernel_size=(9), padding='same', activation='relu')(mid_layers)
mid_layers = tf.keras.layers.MaxPool1D(pool_size=5)(mid_layers)
mid_layers = tf.keras.layers.Dropout(0.5)(mid_layers)
mid_layers = tf.keras.layers.Flatten()(mid_layers)
mid_layers = tf.keras.layers.Dense(64, activation='relu')(mid_layers)
mid_layers = tf.keras.layers.Dense(16, activation='relu')(mid_layers)
output1 = tf.keras.layers.Dense(2, activation='softmax')(mid_layers)
output2 = tf.keras.layers.Dense(2, activation='softmax')(mid_layers)
output3 = tf.keras.layers.Dense(2, activation='softmax')(mid_layers)
output4 =tf.keras.layers.Dense(2, activation='softmax')(mid_layers)
output5 = tf.keras.layers.Dense(2, activation='softmax')(mid_layers)
model = tf.keras.Model(inp_layer,[output1,output2,output3,output4,output5])

# Define custom optimizers
adam = tf.keras.optimizers.Adam(
    learning_rate=0.005,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1,
    amsgrad=False,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=None,
    jit_compile=True,
    name='Adam',
)

## Compile model with metrics
model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
print("CNN model built: ")
model.summary()
```

Figure 7.3.1 - CNN layers and Optimizers

This code is creating a CNN with an input layer followed by an embedding layer with 13 vector sizes follows by a convolution layer with 512 input filters and kernel size 9 ReLU activation function has been used for better optimization followed by a pooling layer followed with flatten layer that follows with 2 dense layers and 5 output layers with SoftMax activation function has been created. Then a custom optimizer has been defined with learning rate and an Adam optimizer has been used. Finally, a model summary has been printed.

```
# model training
class_weights = [{0: 1., 1: 5.},{0: 1., 1: 5.},{0: 1., 1: 5.},{0: 1., 1: 5.},{0: 1., 1: 5.}]

history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs= EPOCHS, # 20 Epochs has been found to be sufficient
                    verbose=1, # 0 = silent, 1 = progress bar, 2 = one line per epoch.
                    validation_data=(x_validate, y_validate),
                    callbacks=[tbCallback, mcp])

# Save history of each epoch
with open('history', 'wb') as file_pi:
    pickle.dump(history.history, file_pi)

# Save the latest model to disk for later use
model.save('model.h5')
```

Figure 7.3.2 - Model building and saving.

Here the model training has been done with 20 iterations/epochs and the model history and model have been saved for later uses like evaluation and deployment. The app.py file screen shot can be found in Appendix C – Implementation (App.py). Further, the whole implementation of the project will be available here on GitHub.

7.4 User interface

The user interface has been moved to Appendix D – User interface.

7.5 Chapter summary

This chapter covered the implementation components of the selected tools, along with a justification for the project. Additionally, it includes visual representations of the implementation snippets of the primary project components.

CHAPTER 8: TESTING

8.1 Chapter Overview

This chapter presents a logical evaluation of Sherlock, which is a significant aspect of the research. The evaluation begins by outlining the objectives and goals of the testing process. It then proceeds to assess all components of Sherlock using relevant metrics, including both functional and non-functional requirements. Finally, the chapter discusses the limitations of the testing process.

8.2 Objectives and Goals of Testing

The primary goal of testing is to ensure that Sherlock works as expected. The following objectives were developed to ensure the achievement of this goal,

1. Test all components of Sherlock to ensure they are functioning properly and producing accurate outputs.
2. Verify that all "Must have" and "Should have" functional requirements outlined in the SRS have been met in Sherlock.
3. Verify that all "Must have" and "Should have" non-functional requirements stated in the SRS have been met in Sherlock.
4. Identify any potential defects or bugs in Sherlock through testing.

8.3 Testing Criteria

In order to organize the testing process two test criteria were introduced as follows,

Criteria	Description
Functional Testing	Focus on the technical and functional aspects of the system to ensure the how the functional requirements of the system have been met.
Structural Testing	Focus on structural and non-functional aspects of the system to ensure how the non-functional requirements if the system have been met.

Table 8.3.1 - Testing criteria

8.4 Model Testing

The original dataset was divided into three parts, namely training, validation, and testing, using an 80:10:10 ratio. All these splits underwent the same data processing and tokenization process before being used for training and evaluation. The training split was used to assess the performance of the model and measure specific evaluation metrics identified in the Evaluating software vulnerability detection. and evaluation was implemented as shown in Figure 8.4.1.

```
# Evaluating the model
for col in range(1,6):

    # Giving names to the columns for better understanding with a dictionary
    vul_name = {
    1: 'CWE-119',
    2: 'CWE-120',
    3: 'CWE-469',
    4: 'CWE-476',
    5: 'CWE-Other',
    }[col]

    print('————————————————————')
    print('Evaluation for',vul_name)
    print('————————————————————')

    # Performance metrics
    print('\nAccuracy: '+ str(sklearn.metrics.accuracy_score(y_true=test.iloc[:,col].to_numpy(), y_pred=pred_test[col-1])))
    print('Precision: '+ str(sklearn.metrics.precision_score(y_true=test.iloc[:,col].to_numpy(), y_pred=pred_test[col-1])))
    print('Recall: '+ str(sklearn.metrics.recall_score(y_true=test.iloc[:,col].to_numpy(), y_pred=pred_test[col-1])))
    print('F1 score: '+ str(sklearn.metrics.f1_score(y_true=test.iloc[:,col].to_numpy(), y_pred=pred_test[col-1])))
    print('Precision-Recall AUC: '+ str(sklearn.metrics.average_precision_score(y_true=test.iloc[:,col].to_numpy(), y_score=predicted[col-1][:,1])))
    print('AUC: '+ str(sklearn.metrics.roc_auc_score(y_true=test.iloc[:,col].to_numpy(), y_score=predicted[col-1][:,1])))
    print('————————————————————')

    print('————————————————————')
    print('Classification Report for',vul_name)
    print('————————————————————')

    # Classification report
    print(sklearn.metrics.classification_report(y_true=test.iloc[:,col].to_numpy(), y_pred=pred_test[col-1]))

    print('————————————————————')
    print('Confusion Matrix for',vul_name)
    print('————————————————————')

    # Confusion matrix
    confusion = sklearn.metrics.confusion_matrix(y_true=test.iloc[:,col].to_numpy(), y_pred=pred_test[col-1])
    group_names = ['TN','FP','FN','TP']
    group_counts = ['{0:0.0f}'.format(value) for value in confusion.flatten()]
    group_percentages = ['{0:.2%}'.format(value) for value in confusion.flatten()/np.sum(confusion)]
    labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(group_names,group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(confusion, annot=labels, fmt='', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix for '+vul_name)
    plt.xticks([0.5,1.5],['Non-Vulnerable','Vulnerable'])
    plt.yticks([0.5,1.5],['Non-Vulnerable','Vulnerable'])
    plt.show()

    print('————————————————————')
    print('ROC curve for',vul_name)
    print('————————————————————')

    # ROC curve
    fpr, tpr, thresholds = sklearn.metrics.roc_curve(y_true=test.iloc[:,col].to_numpy(), y_score=predicted[col-1][:,1])
    plt.plot(fpr, tpr, label='ROC curve')
    plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC curve')
    plt.legend(loc='best')
    plt.show()

```

Figure 8.4.1 - Code snippet for model testing

8.5 Confusion Matrix

The performance of the model in classifying each vulnerability has been assessed using a confusion matrix for each vulnerability, as shown in Figure 8.5.1.

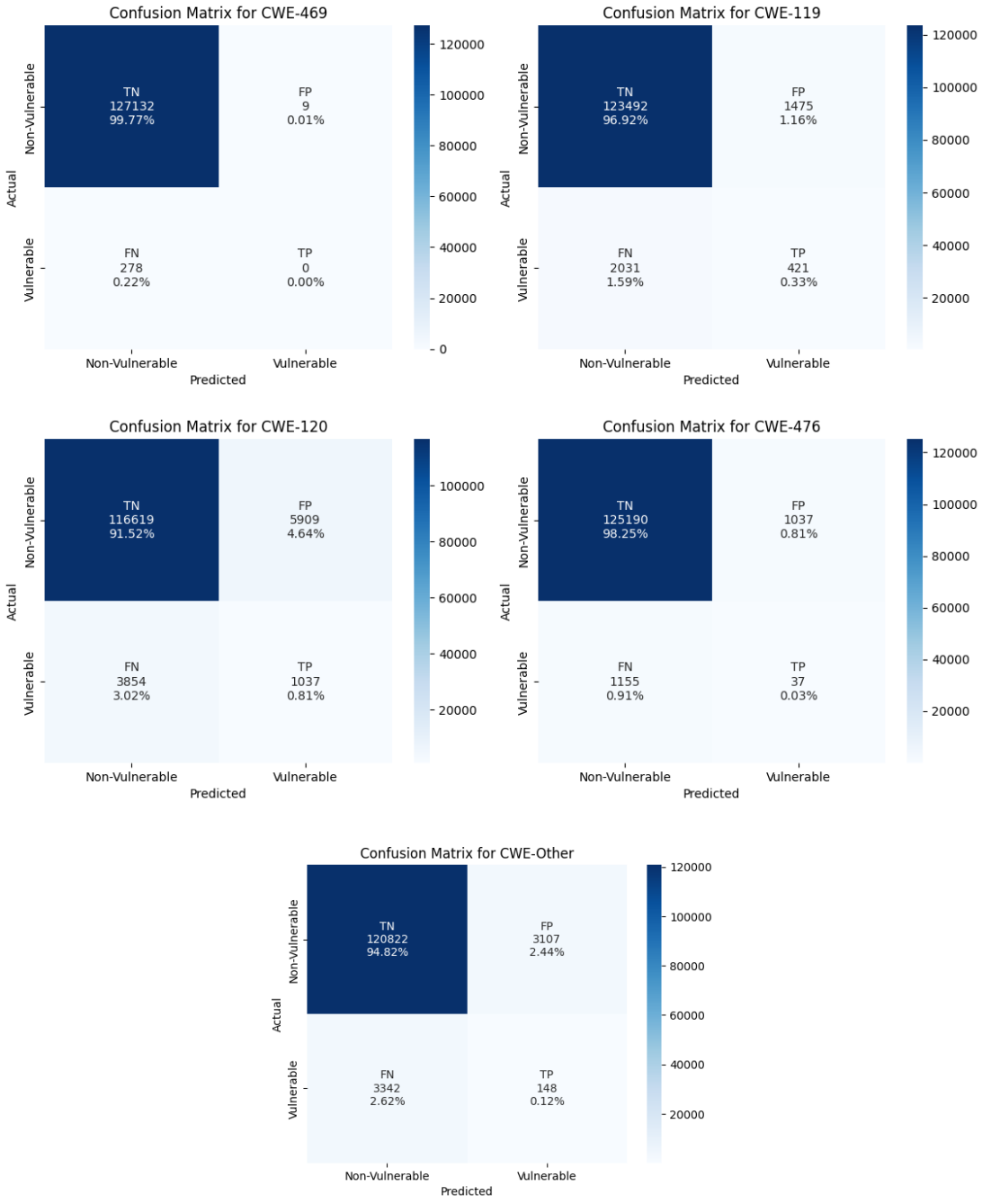


Figure 8.5.1 - Confusion Matrix for each type of vulnerabilities (Self-composed)

Overall, the model appears to be effective in classifying true negatives. However, the main goal of this model is to identify true positive cases, and its performance in classifying true positives for CWE-469 and CWE-476 is not as strong. The reasons for these issues will be discussed in coming chapters. It should be noted that the major cause of these issues is the unbalanced nature of the dataset.

8.6 Performance metrics

In addition to the confusion matrix, Accuracy, precision, recall, F1 score, and AUC values were chosen as performance metrics based on the literature review. The model was evaluated for each CWE vulnerability using these performance metrics, and the results are presented in the Table 8.6.1.

Metric / CWE	Accuracy	Precision	Recall	F1 Score	AUC
CWE-199	0.97	0.22	0.17	0.19	0.81
CWE-120	0.92	0.15	0.21	0.18	0.72
CWE-469	0.99	0.0	0.0	0.0	0.83
CWE-476	0.98	0.03	0.03	0.03	0.54
CWE-Other	0.95	0.04	0.04	0.04	0.67

Table 8.6.1 - Model performance for each vulnerability

The high accuracy and AUC values suggest that the overall model performance is good in terms of various vulnerabilities. However, the low precision and recall values indicate a high false positive and false negative rate. The evaluation indicates that the model's performance is good and reliable for CWE-199 and CWE-120. However, its performance is poor, particularly for CWE-469 and CWE-476. This suggests that the model needs to be improved in terms of its ability to correctly

classify instances across all categories. The detailed classification report and ROC curves can be found in the Appendix C – Test results.

8.7 Benchmarking

As previously mentioned in the section 2.8 , due to the lack of a standard dataset, it is no longer appropriate to perform benchmarking. However, the model's metrics have been compared to previous works for detecting CWE-199 to provide a better comparative view.

Metric / Model	Precision	Recall	F1 Score
Code2vec + MLP (Baseline)	0.06	0.87	0.12
Sherlock (Ours)	0.22	0.17	0.19

Table 8.7.1 - Benchmarking with baseline models

Baseline model was taken based as suggested by Bilgin et al., (2020) and even though the sherlock is capable of detecting multiple vulnerabilities the performance were compared only for CWE-199 since the baseline model cannot do so.

8.8 Functional Testing

Test was performed for all the implemented functional requirements identified from the Functional requirements section as displayed in the Table 8.8.1.

Id	FR Id	User action	Expected outcome	Actual outcome	Result
1	FR 3	Pasting or typing the required C/C++ function to be tested for vulnerability.	Provided function code should be displayed in the text area.	Provided function code was displayed in the text area.	Passed

2	FR 1	Press the “Predict” button	System should process the given source code and display the prediction.	System processed the given source code and display the prediction.	Passed
3	FR 4, FR 7	Try with a function which has purposefully created vulnerability.	System should display an ideal prediction in a meaningful way with confidence level.	System displayed an ideal prediction in a meaningful way with confidence level.	Passed
4	FR 2	Trying with various function in the system.	System should display different type of vulnerabilities.	System displayed different type of vulnerabilities.	Passed
5	FR 8	Check the provided results for context	System should provide CWE type and the name of the verbality along with context.	System was provided CWE type and the name of the verbality along with context.	Passed
6	FR 6	Select different model versions and try predicting	System should be able to predict with different models and provide results	System was e able to predict with different models and provide results	Passed

Table 8.8.1- Functional testing

8.9 Module and Integration Testing

The modules of Sherlock have been tested as shown in the Table 8.9.1

Module	Input	Expected outcome	Actual outcome	Results
Tokenizer module	Function source code	Tokenized sequence of tokens	Tokenized sequence of tokens	Passed
Vulnerability detection module	Venerable code	Detected vulnerability and confidence level	Detected vulnerability and confidence level	Passed

Table 8.9.1 - Module Testing

8.10 Non-Functional Testing

Non-functional requirements gathered from the Non-Functional requirements section has been tested as discussed below.

8.11 Accuracy Testing

The accuracy of Sherlock's ability to detect software vulnerabilities was clearly evaluated using relevant metrics in sections 8.6 and 8.7

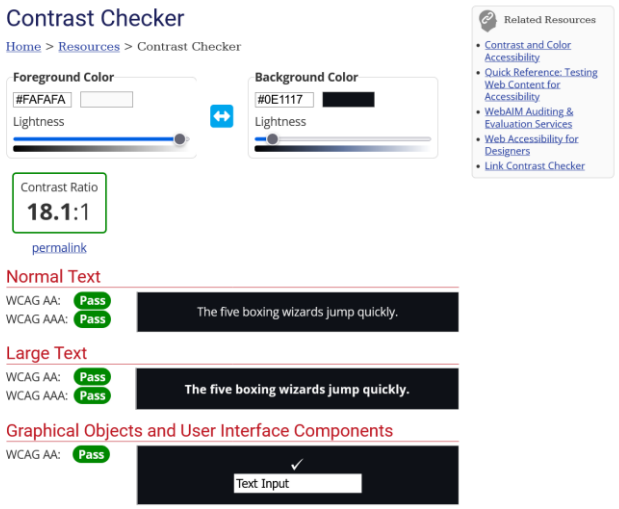
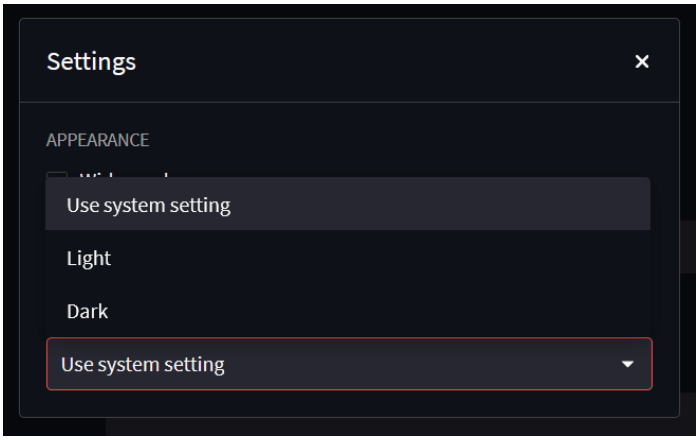
8.12 Reliability Testing

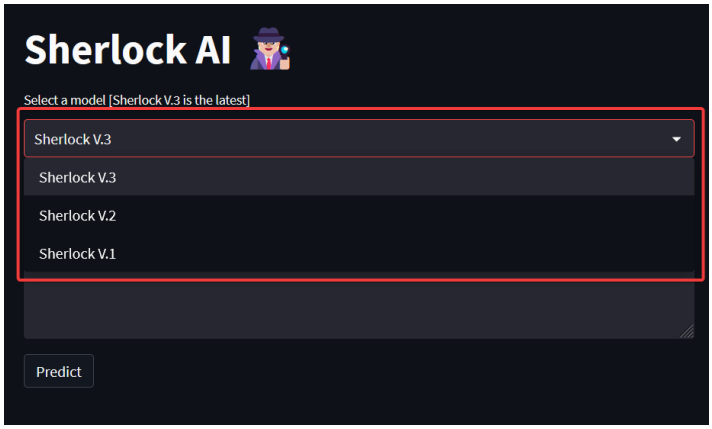
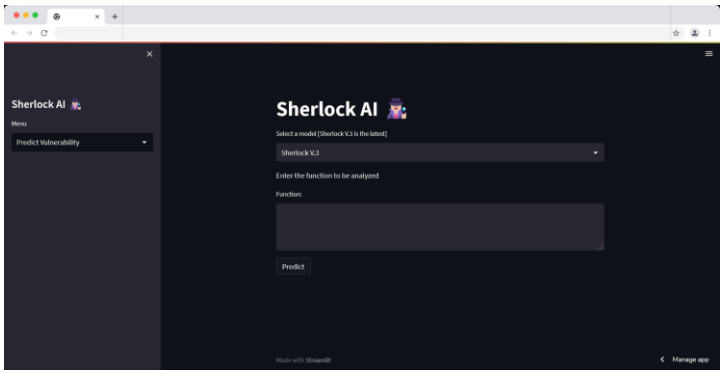
The reliability of a classification system like Sherlock is primarily evaluated using the values of the confusion matrix, and a detailed discussion of the confusion matrix can be found in section 8.5

8.13 User experience Testing

User experience has been tested with usability heuristics proposed by Jakob Nielsen, (1994) and the discussion is presented with proofs in Table 8.13.1.

Heuristics	Discussion	Proof

Consistency and standards	The web application has passed the accessibility standard test of W3.org accepted evaluation test which can clearly show that the website is accessible for everyone and developed which accessibility in mind. Further the consistency has been maintained	
User control and freedom	The user has been granted with the ability to customize the system with their own of their own choice of theming.	

Flexibility and efficiency of use	The system allows the user to choose their preferred model version which will give more flexibility and choice freedom based on their preferences	
Aesthetic and minimalist design	The user interface is designed and developed to provide a minimalist user interface which is easy to understand and provide a good user interface with material elements.	
Recognition rather than recall	All buttons and options were made clearly visible and labelled for the user to easily	

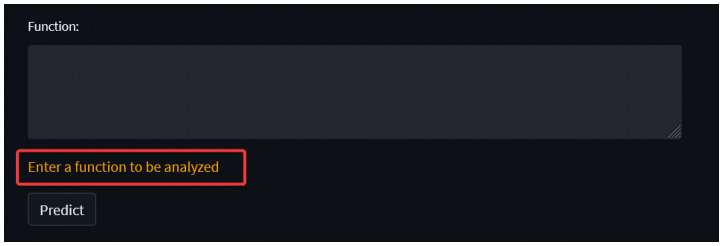
	recognize the options.	
Error prevention	Good error handling is preventing the error from occurring. So, proper error prevention methods were used through out the system to ensure this.	

Table 8.13.1 - UX testing with Jakob Nielsen's heuristics.

8.14 Limitations of the testing process

The primary limitation of the testing is the absence of a proper benchmarking model and dataset. This was addressed by using a baseline model proposed by Bilgin et al., (2020). However, the baseline model can only predict one vulnerability at a time, which may still be considered a limitation. The lack of a standard dataset also restricted the author from performing dataset benchmarking. Additionally, some of the secondary non-functional requirements were not tested due to their less significant and subjective nature.

8.15 Chapter Summary

As the saying goes, numbers and results speak better than words. The evaluation of Sherlock demonstrated that it is a promising solution for the addressed problem. However, a good evaluation also identifies areas for improvement. The evaluation of the model showed that although its performance is reliable for most vulnerabilities, it struggles with certain ones. Through the evaluation, the author was able to identify the cause of this issue, which was an unbalanced dataset.

Despite applying known techniques such as random sampling, resampling etc. there is still room for improvement, which will be addressed in future work.

CHAPTER 9: EVALUATION

9.1 Chapter overview

This chapter presents a logical evaluation of Sherlock, starting with the approaches taken and the evaluation criteria. It then includes a critical self-evaluation by the author and thematic analysis of evaluations provided by evaluators of specific categories. Finally, the chapter discusses the limitations of the evaluation process.

9.2 Evaluation methodology and approach

Sherlock is composed of a machine learning model that requires a quantitative evaluation and a user interface that requires a qualitative evaluation. As such, both quantitative and qualitative evaluations were conducted. Quantitative evaluations were carried out using relevant metrics, while qualitative evaluations were performed through thematic analysis of expert and user interviews, as well as survey results.

9.3 Evaluation criteria

Selected evaluation criteria and their purposes were discussed in the Table 1.11.1.

Criteria	Purpose
Novelty	The novelty of the problem and the technology used to address it will be evaluated.
Challenge and scope	The research scope will be evaluated to determine whether it presents enough challenges.
Technology selection and development approaches	The appropriateness of the selection of technologies, development tools, and practices will be evaluated to ensure they meet industry standards.

Vulnerabilities detection and selection	The reliability of vulnerability detections with experts will be evaluated, as well as the suitability of the selected vulnerabilities in terms of their severity.
Performance metrics	The selection of metrics and their corresponding results will be evaluated.
User interface and User experience	The user interface will be evaluated to determine whether it is sufficient and provides an optimal user experience for all use cases.
Future direction and suggestions	The feedback received for Sherlock will be evaluated to identify areas for improvement and determine potential directions for future development and commercialization

Table 9.3.1 - Evolution criteria

9.4 Self-evaluation

Self-evaluation of Sherlock has been discussed in Table 9.4.1

Criteria	**Evaluation**
Novelty	Software vulnerabilities have been a longstanding issue, and current techniques have proven to be insufficient, leading to massive negative impacts for individuals and organizations. Traditional code analysis approaches are limited in detecting multiple vulnerabilities prior to deployment, calling for a new data-driven approach. While data-driven approaches such as AI are widely used in other domains, they are relatively new to this field. Sherlock's novelty lies in its ability to detect multiple vulnerabilities in the source code (functions) and achieve this through a novel deep learning model.

Challenge and scope	Considering the fact that only a few previous works are addressing this problem of detecting software vulnerability with deep learning even fewer of them are addressing multiple vulnerability detection. Data scarcity is another big hurdle for this issue since the approach is totally relay on data availability. As aforementioned there is enough challenges in achieving the scope of the research.
Technology selection and development approaches	To the best of the author's knowledge and with their one-year industrial experience, the most current tech stack and industry-standard practices were utilized in the development of Sherlock. However, an expert evaluation may still be necessary for this aspect.
Vulnerabilities detection and selection	To the authors best knowledge about the vulnerabilities which was gained along with the research the model is quite good at detecting most of the selected vulnerabilities (CEW-119, CWE-120 and CWE-other) the model performance is not that excellent in some vulnerabilities (CWE- 469) the cause of this issue is data scarcity which is also clearly identified yet due to time constraints and expert suggestions it has been suggested as a future work.
Performance metrics	The model performed well, achieving over 90% accuracy in detecting various vulnerabilities. It demonstrated strong classification of true positive and true negative cases, with low rates of false positives and false negatives, indicating good performance. Additionally, the recall and precision values were generally good and reliable for most vulnerabilities. However, the model struggled with certain vulnerability types due to a lack of data.
Overall	Overall, Sherlock was made possible by the author’s continuous efforts and commitment. And it is clearly a reliable solution to

	identified problems. Further, it has enough challenges, scope and contribution. Also clearly suggest and create a new research area for future researchers.

Table 9.4.1 - Self-evaluation

9.5 Selection of the evaluators

The evaluators for the project were categorized as displayed in Table 9.5.1 in order to narrow down the evaluation.

Category Id	Description
CAT1	Domain experts: Cyber security experts, developers who promotes secure systems and quality checker were considered.
CAT2	Technical experts: Previous graduates and researchers who did researches in similar domain, machine learning experts, data engineers and lecturer were considered.
CAT3	Targeted users: Developers, Software testers and other users were considered.

Table 9.5.1 - Categories of evaluators

9.6 Evaluation result

The thematic analysis findings for each criterion and categories were presented in Table 9.6.1.

Criteria	Category Id	Theme	Summarized opinion
Novelty	CAT1, CAT2	Novelty in the problem and research domain.	While the issue of detecting multiple vulnerabilities prior to deployment is not new, state-of-the-art techniques have proven to be inefficient. Therefore, the system

			provided in this research, which is capable of detecting multiple vulnerabilities prior to deployment, represents a novel contribution to the field. Although deep learning has been applied in various other domains, its usage in this particular problem domain is new. Moreover, the proposed approach of using a five-output layer to detect five different types of vulnerability is also a novel contribution.
Challenge and scope	CAT1, CAT2	Challenges in achieving scope	The primary challenge for this research is the scarcity of data, as there is no standard dataset available, making it difficult to apply deep learning techniques. Additionally, interpreting the results is a challenge, as incorrect interpretation could render the findings meaningless. Despite these challenges, the research successfully addressed these issues and achieved the proposed scope of work. Any elements that fell outside of the scope were also clearly identified.
Vulnerabilities detection and selection	CAT1	Severity of addressed vulnerability and	The selected vulnerabilities for this research, namely CWE-199, CWE-120, CWE-469, CWE-476, and CWE-Other (which contains five or more common vulnerabilities), are ideal and standard

		correctness of system.	choices. The model performs well in detecting most of these vulnerabilities. However, the performance is not as satisfactory for CWE-469 and CWE-476.
Technology selection and development approaches	CAT2	Tech stack selection and coding practices	The ideal tech stack was selected for this problem, and good error handling practices were implemented throughout the project. Additionally, version control was maintained, with a separate branching for deployment, which is a good practice in software development. Streamlit is a new and emerging platform that is not yet widely used in the industry, Thus, the knowledge gained on this emerging technology is valuable.
Performance metrics	CAT2	Classification metrics and performance	Since this is a classification problem all required metrics like confusion matrix, accuracy, recall, precision and F1 score was provided to showcase the performance.
User interface and User experience	CAT3	UI/UX and user freedom	Minimal and user-friendly interface, with consistent theming and labeling. The users have the freedom to access older versions of the models and can also select different themes, which is useful. The input fields are designed to prevent errors, which is commendable. However, some evaluators have found the UI to be oversimplified.

Future direction and suggestions	CAT1, CAT2, CAT3	Future works and improvements	The unbalanced nature of the dataset has caused issues in detecting some vulnerabilities. Future work could focus on improving the dataset's balance to enhance the model's performance. Additionally, the model could be converted into an IDE plugin for more appropriate use and added versatility in addition to the current web application. Multi-language support, such as Python, could be added to improve the model's applicability. Furthermore, the model's effectiveness could be further improved by checking for more vulnerabilities and experimenting with NLP and ensemble approaches

Table 9.6.1 - Evaluation results

The application was deployed and hosted in Streamlit cloud for evaluation here.

9.7 Limitations of evaluation

One of the main limitations in evaluating the project is the lack of available experts who have experience with similar projects. Despite the author's attempts to contact experts in each of the selected categories, many were unable to participate due to their busy schedules. As a result, the number of evaluators for the project was limited. However, the author was able to mitigate this limitation by continuously seeking feedback from the available evaluators to improve the project.

9.8 Evaluation of functional requirements and non-functional requirements

The functional requirements were evaluated in section 8.8 and the non-functional requirements were evaluated in section 8.10 And all these requirements were successfully implemented as proposed.

9.9 Chapter summary

This chapter focuses on the evaluation of Sherlock, a software system, and presents the approaches and criteria used to evaluate it, as well as the author's own evaluation. The chapter also categorizes the evaluators and presents their evaluation results. Finally, the chapter discusses the limitations of the evaluation and provides clear guidance on where to find functional and non-functional requirements.

CHAPTER 10: CONCLUSION

10.1 Chapter overview

This chapter is a conclusion for this PSPD which analyses the deviation from the proposal and initial test results were included within also a demo has been attached for a better understanding of the problem and system.

10.2 Achievements of Research Aims & Objectives

The aim of the research is to design, develop and evaluate a novel software vulnerability detection system with artificial intelligence.

The aim of the research was successfully achieved through the design, development, and evaluation of a deep learning software vulnerability detection system that can detect multiple vulnerabilities from source code. To achieve this, the author established primary research objectives and learning outcome for the module, as shown in the Table 10.2.1.

Objectives	**Learning outcomes**	**Status**
Problem Identification	LO1, LO2	Achieved
Literature Review	LO1, LO4, LO8	Achieved
Methodology Selection and SLEP Framework	LO2, LO6, LO8	Achieved
Requirement Elicitation	LO1, LO3, LO5, LO8	Achieved
Design	LO1, LO5, LO8	Achieved
Implementation	LO1, LO5, LO7, LO8	Achieved
Evaluation	LO1, LO5, LO8	Achieved

Table 10.2.1 - Achievement of objectives

10.3 Utilization of knowledge from the course

The module offered through the courses and knowledge that the author used for the completion of this project was discussed in Table 10.3.1

Modules	Knowledge
SDGP	This module provided valuable learning opportunities in research and development, as well as in testing and design, especially in the context of group project work.
Machine learning and Data mining & Applied AI	These modules provided a fundamental understanding of data science, introducing key concepts in machine learning algorithms and evaluation metrics. Through these modules, you were able to acquire knowledge and skills in these areas.
Cyber security	The module introduced the fundamental concepts of secure software development and emphasized the potential impact of vulnerabilities. Through hands-on experience, author gained a deeper appreciation for the importance of secure software and the need to prioritize security throughout the development process.
PP1, PP2 & OOP	These modules served as an introduction to programming, Integrated Development Environments (IDEs), programming principles, and test cases. They provided a solid foundation for the development.
Web design and development	This module introduced the fundamental concepts of web development, as well as the initial idea of User Interface and User Experience testing.
Usability testing	This module taught statistical approaches to conducting surveys and emphasized the importance of ethical data normalization techniques.

Table 10.3.1 - Utilization of knowledge from the course

10.4 Use of existing skills

The author's existing research and development skills were instrumental in the successful completion of this project. In addition, the machine learning and deep learning skills acquired from the modules, as well as a personal curiosity and interest, proved to be valuable assets. Furthermore, the author's experience in writing blogs on their website was leveraged in preparing the project reports.

10.5 Use of new skills

The following skills were obtained throughout this project,

- **Knowledge in software vulnerabilities domain**: This research provided the author with a wealth of knowledge about software vulnerabilities, their impact, severe vulnerabilities, existing technologies in use, and more.
- **Developing a deep learning model**: The author gained a comprehensive understanding of various deep learning approaches and architectures and learned how to select models to address specific problems. Furthermore, the author learned how to benchmark models, evaluate them using different metrics, and conduct testing.
- **Deploying models with Streamlit**: The author also learned about Streamlit, an emerging technology used to create Graphical User Interfaces (GUIs) integrated with deep learning models, and how to host a web application in Streamlit.

10.6 Achievement of learning outcomes

The learning outcomes of the Final year project module, how it was achieved, and their status is discussed in Table 10.2.1.

LO	How it was achieved	Status
LO1	A large enough problem was identified and tackled with appropriate methods, tools and techniques.	Achieved

LO2	The project was managed with an ideal project management and scheduling, primary schedule was provided by the module team.	Achieved
LO3	Appropriate Requirement elicitation was done with relevant instruments.	Achieved
LO4	Mediums like literature review, interviews and brainstorming was used to gather information and findings were gathered.	Achieved
LO5, LO7, L08	Autonomous literature review, documentation and development was done till the last moments of research and submitted all working deliverables with supervisors' approval. And all deliverables were critically evaluated.	Achieved
LO6	SLEP framework was ensured in all aspects of research	Achieved
LO9	Preparation for viva has been done and having confidence on the submitted work.	Pending

Table 10.6.1 - Achievement of learning outcomes

10.7 Problems and challenges faced

Some major challenges faced while doing this research and the mitigation process to overcome the research has been discussed in Table 10.7.1.

Challenge	**Mitigation**
Data limitation and scarcity in open datasets, obtained dataset was unbalanced.	The only reliable private Draper VDISC dataset was obtained by requesting access and license from OSF community and resampling and other techniques were tried to overcome unbalance this challenge.

Hardware limitations and GPU hungry tasks demands a more power powerful device than the author's one.	To leverage free GPU resources and powerful virtual environments, cloud-based platforms such as Google Colab, GitHub Code Spaces, and Streamlit Cloud were utilized in this project.
Have to rerun the collab file repeatedly to train the model and it will automatically discard when let inactive.	To ensure efficient training and keep track of the best model at each epoch, callback points and continuous training histories were used in this project. This allowed the model to resume training from where it left off in case of interruptions or errors, and to save the best model achieved during training.
Have to run the model each time when we have to run the system.	To facilitate model deployment, the trained model was saved in a pickle file format and was used for deployment instead of re-running the model. This approach helped to save time and resources, and allowed for faster and more efficient deployment of the model.
Evaluators won't be able to check the app running in local host.	The deployed model was hosted on Streamlit Cloud, making it accessible to anyone who wanted to try and use it. This allowed for easy deployment and sharing of the model with others

Table 10.7.1 - Challenges and mitigation

10.8 Deviations

There was a significant deviation from the original schedule (Appendix B – Ganntchart) projected for this project and using an agile approach the author manages to overcome this and submit the deliverables on time.

10.9 Limitations of the research

The following are the identified limitations of Sherlock,

- Even though the model performance is good for most of the vulnerabilities, model performance is not that good for some vulnerabilities which can be considered as a limitation.
- Including the above performance issue and some other issues were caused by the unbalanced dataset even after the approaches tried to balance the dataset this characteristic remains so; this can be considered as a limitation of research.
- System is limited to C and C++ functions.

10.10 Future enhancements

Sherlock opened a whole new path for data-driven multiple vulnerability detection and the potential it creates for future research can be addressing the following,

- Creating a balanced labelled dataset and using that to train the proposed model can bring out more robust and ground braking performance which can outperform the proposed model.
- Expanding the coverage of vulnerabilities could be a valuable avenue for future research.
- Training the model on source code from other programming languages, such as Python, and creating a separate output split could enhance the model's coverage to incorporate other programming languages.
- Ensemble the model with and NLP component might be a promising new landscape to check for.
- Additionally applying this model for other problems can also be a future research area.

10.11 Achievement of the contribution to body of knowledge

The author was able to achieve all the contributions mentioned in **Contribution to the body of knowledge section** with a system to detect multiple vulnerabilities prior to deployment from source code which utilizes a deep neural network with five output splits.

10.12 Concluding remarks

Sherlock is a product of one-year consistent efforts, dedication of the author. Initially, this was just a conceptual idea now seeing it as product it feels good. Sherlock is proof that data driven approaches can be used to detect multiple vulnerabilities in software and it can be more reliable and effective than the state of art techniques. Primarily the contribution of Sherlock relay on its ability to detect multiple vulnerabilities in source code prior to deployment and using a deep neural network with five output splits. Sherlock opens a new research space for future researchers which will continue the legacy of Sherlock. Finally, the name Sherlock is inspired by a fictional detective who stops crimes even before they occur, similarly, Sherlock will stop vulnerabilities even before they do the damage.

REFERENCES

Aiyer, B. et al. (2022). New survey reveals $2 trillion market opportunity for cybersecurity technology and service providers.

Bilgin, Z. et al. (2020). Vulnerability Prediction From Source Code Using Machine Learning. *IEEE Access*, 8, 150672–150684. Available from https://doi.org/10.1109/ACCESS.2020.3016774.

Coker, J. (2023). The LastPass Breaches: Password Managers in the Spotlight. *Infosecurity Magazine*. Available from https://www.infosecurity-magazine.com/news-features/lastpass-breaches-password/ [Accessed 28 April 2023].

Coulter, R. et al. (2020). Data-Driven Cyber Security in Perspective—Intelligent Traffic Analysis. *IEEE Transactions on Cybernetics*, 50 (7), 3081–3093. Available from https://doi.org/10.1109/TCYB.2019.2940940.

Cowan, C. et al. (1998). StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. *USENIX*.

Dempsey, K. et al. (2019). *Automation Support for Security Control Assessments: Software Vulnerability Management*. Available from https://doi.org/10.6028/NIST.IR.8011-4-draft [Accessed 31 October 2022].

Ghaffarian, S.M. and Shahriari, H.R. (2018). Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. *ACM Computing Surveys*, 50 (4), 1–36. Available from https://doi.org/10.1145/3092566.

Hanif, H. et al. (2021). The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. *Journal of Network and Computer Applications*, 179, 103009. Available from https://doi.org/10.1016/j.jnca.2021.103009.

Heartbleed Bug. (2020). Available from https://heartbleed.com/ [Accessed 31 October 2022].

Hu, Y. (2019). *A Framework for Using Deep Learning to Detect Software Vulnerabilities*. Available from http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-164112 [Accessed 9 December 2022].

Jakob Nielsen. (1994). 10 Usability Heuristics for User Interface Design. *Nielsen Norman Group*. Available from https://www.nngroup.com/articles/ten-usability-heuristics/ [Accessed 9 May 2023].

Li, Z. et al. (2018). VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. *Proceedings 2018 Network and Distributed System Security Symposium*. 2018. Available from https://doi.org/10.14722/ndss.2018.23158 [Accessed 1 May 2023].

Li, Z. et al. (2022). SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, 19 (4), 2244–2258. Available from https://doi.org/10.1109/TDSC.2021.3051525.

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.

Newsome, J. and Song, D. (2005). Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. 43.

Pewny, J. et al. (2014). Leveraging semantic signatures for bug search in binary programs. *Proceedings of the 30th Annual Computer Security Applications Conference*. ACSAC '14. 8 December 2014. New York, NY, USA: Association for Computing Machinery, 406–415. Available from https://doi.org/10.1145/2664243.2664269 [Accessed 31 October 2022].

Portokalidis, G., Slowinska, A. and Bos, H. (2006). Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. *ACM SIGOPS Operating Systems Review*, 40 (4), 15–27. Available from https://doi.org/10.1145/1218063.1217938.

Russell, R.L. et al. (2018). Automated Vulnerability Detection in Source Code Using Deep Representation Learning. Available from https://doi.org/10.48550/arXiv.1807.04320 [Accessed 7 November 2022].

Ryan. (2022). Project Zero: The More You Know, The More You Know You Don't Know. *Project Zero*. Available from https://googleprojectzero.blogspot.com/2022/04/the-more-you-know-more-you-know-you.html [Accessed 30 October 2022].

Sarker, I.H. et al. (2020). Cybersecurity data science: an overview from machine learning perspective. *Journal of Big Data*, 7 (1), 41. Available from https://doi.org/10.1186/s40537-020-00318-5.

Singh, K., Grover, S.S. and Kumar, R.K. (2022). Cyber Security Vulnerability Detection Using Natural Language Processing. *2022 IEEE World AI IoT Congress (AIIoT)*. 6 June 2022. Seattle, WA, USA: IEEE, 174–178. Available from https://doi.org/10.1109/AIIoT54504.2022.9817336 [Accessed 29 September 2022].

Skybox Security. (2022). Vulnerability and Threat Trends Report 2022. *Skybox Security*. Available from https://www.skyboxsecurity.com/resources/report/vulnerability-threat-trends-report-2022/ [Accessed 27 April 2023].

Sonnekalb, T., Heinze, T.S. and Mäder, P. (2021). Deep security analysis of program code. *Empirical Software Engineering*, 27 (1), 2. Available from https://doi.org/10.1007/s10664-021-10029-x.

State of Cybersecurity Report 2021 | 4th Annual Report | Accenture. (2021). Available from https://www.accenture.com/us-en/insights/security/invest-cyber-resilience [Accessed 31 October 2022].

Sun, N. et al. (2019). Data-Driven Cybersecurity Incident Prediction: A Survey. *IEEE Communications Surveys & Tutorials*, 21 (2), 1744–1772. Available from https://doi.org/10.1109/COMST.2018.2885561.

Tomaschek, A. (2023). LastPass Issues Update on Data Breach, But Users Should Still Change Passwords. *CNET*. Available from https://www.cnet.com/tech/services-and-software/lastpass-issues-update-on-data-breach-but-users-should-still-change-passwords/ [Accessed 28 April 2023].

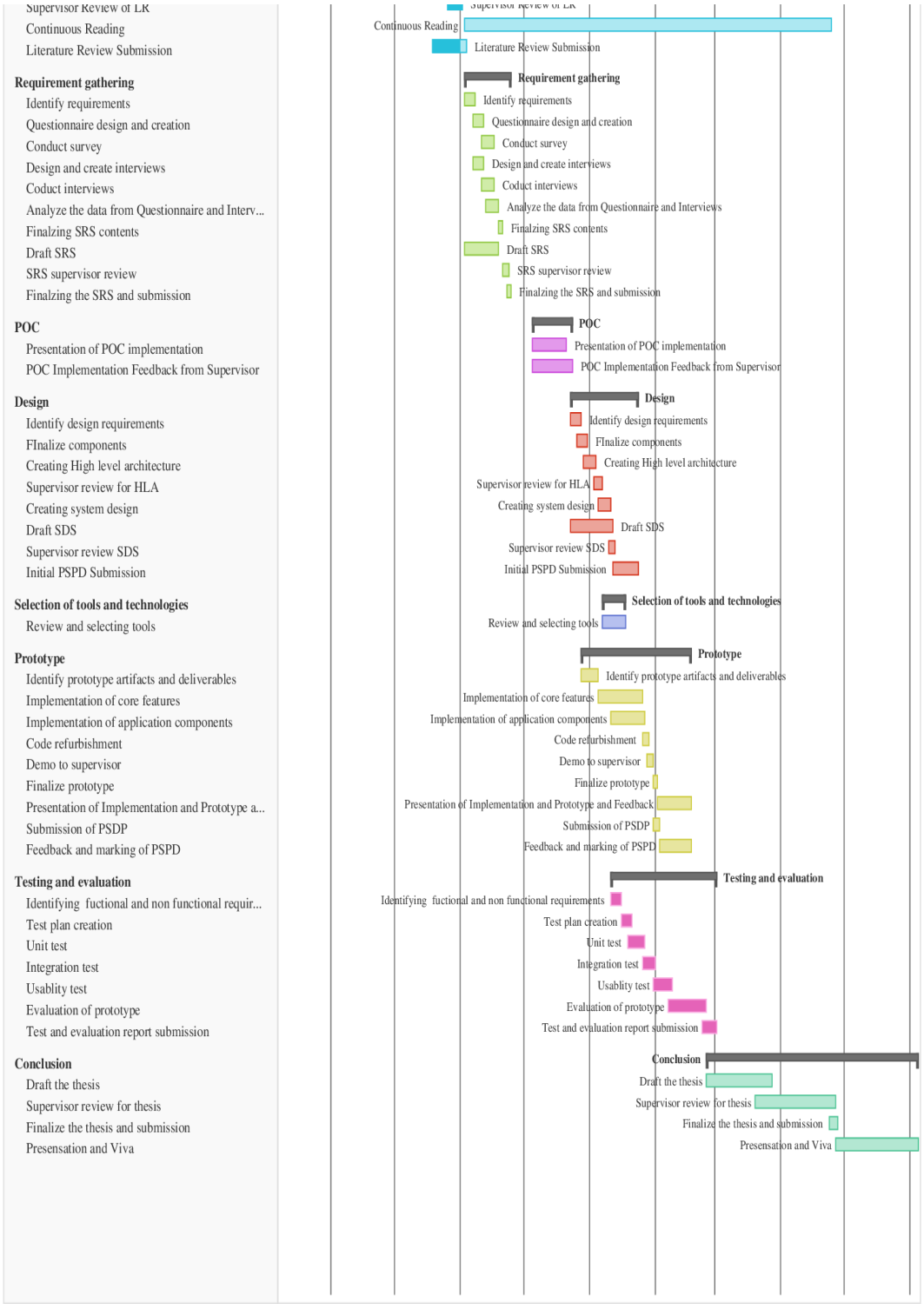
University of Westminster Code of Practice Governing the Ethical Conduct of Research 2020-21. (2021).

Yamaguchi, F. et al. (2014). Modeling and Discovering Vulnerabilities with Code Property Graphs. *2014 IEEE Symposium on Security and Privacy*. May 2014. San Jose, CA: IEEE, 590–604. Available from https://doi.org/10.1109/SP.2014.44 [Accessed 31 October 2022].

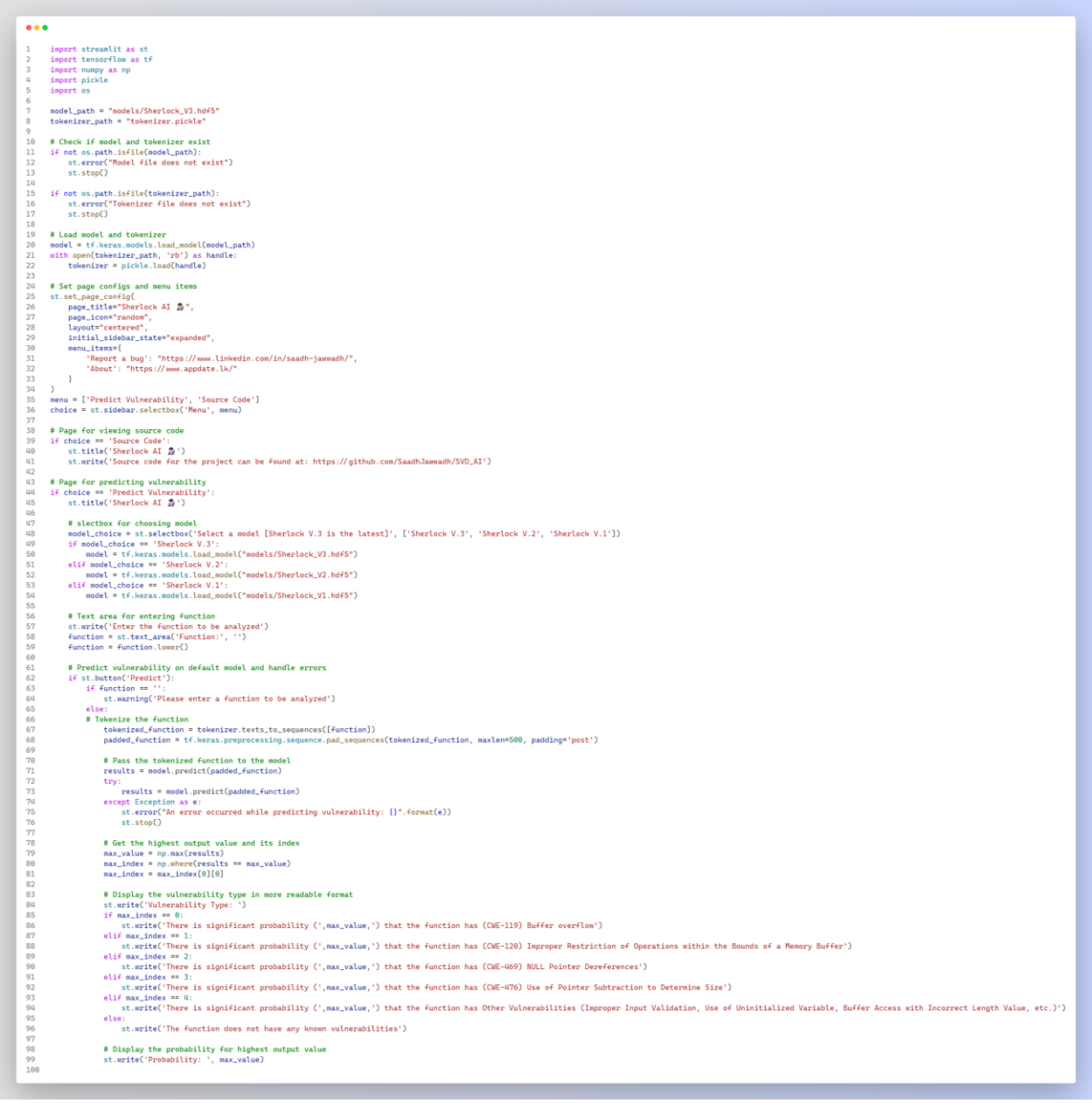
Zero Day Initiative. (2023). Zero Day Initiative — The April 2023 Security Update Review. *Zero Day Initiative*. Available from https://www.thezdi.com/blog/2023/4/11/the-april-2023-security-update-review [Accessed 27 April 2023].

APPENDIX A – CONCEPT MAP

APPENDIX B – GANNTCHART

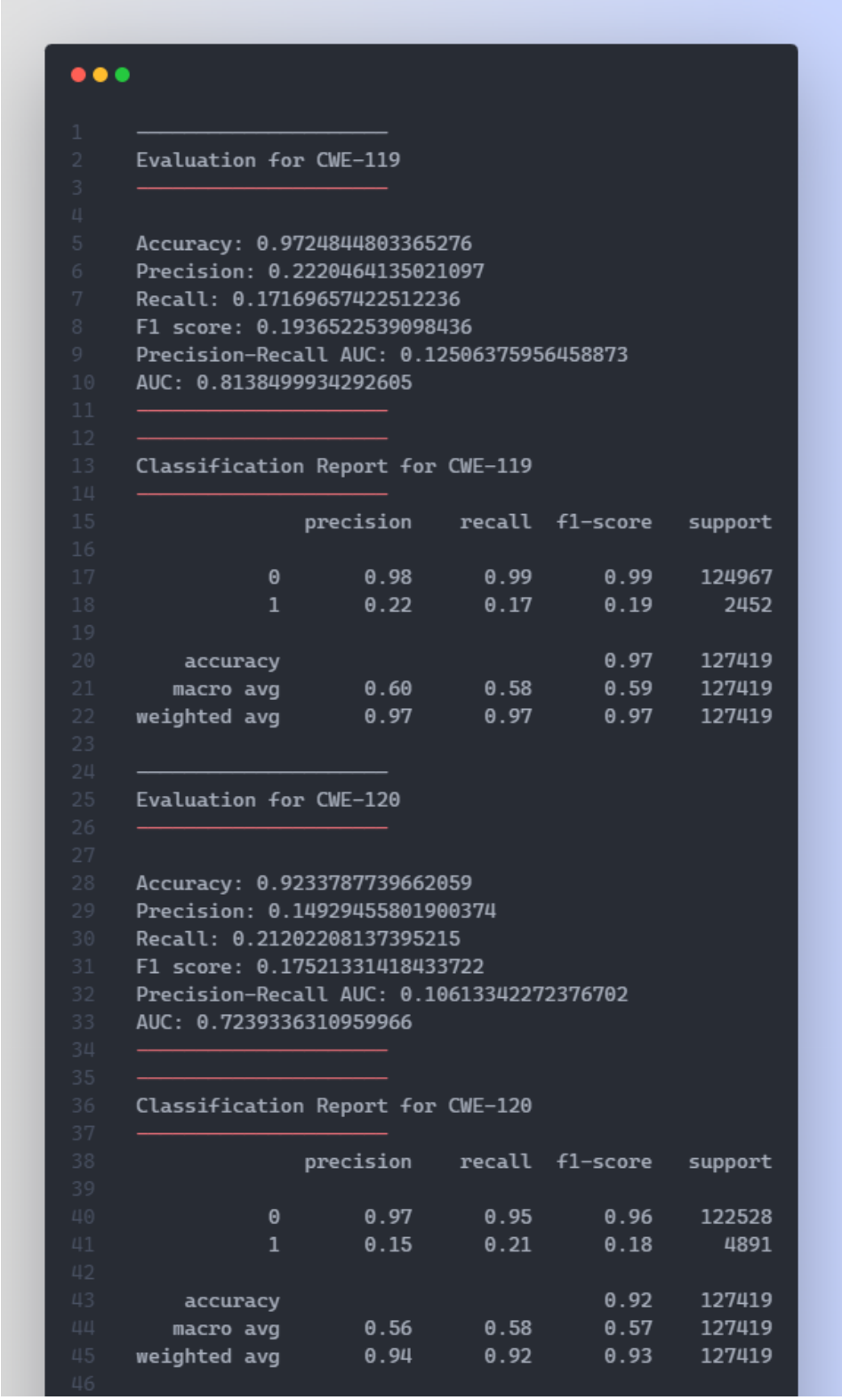


APPENDIX C – IMPLEMENTATION (APP.PY)



```python
import streamlit as st
import tensorflow as tf
import numpy as np
import pickle
import os

model_path = "models/Sherlock_V3.hdf5"
tokenizer_path = "tokenizer.pickle"

# Check if model and tokenizer exist
if not os.path.isfile(model_path):
    st.error("Model file does not exist")
    st.stop()

if not os.path.isfile(tokenizer_path):
    st.error("Tokenizer file does not exist")
    st.stop()

# Load model and tokenizer
model = tf.keras.models.load_model(model_path)
with open(tokenizer_path, 'rb') as handle:
    tokenizer = pickle.load(handle)

# Set page configs and menu items
st.set_page_config(
    page_title="Sherlock AI 🕵",
    page_icon="random",
    layout="centered",
    initial_sidebar_state="expanded",
    menu_items={
        'Report a bug': "https://www.linkedin.com/in/saadh-jawwadh/",
        'About': "https://www.appdate.lk/"
    }
)
menu = ['Predict Vulnerability', 'Source Code']
choice = st.sidebar.selectbox('Menu', menu)

# Page for viewing source code
if choice == 'Source Code':
    st.title('Sherlock AI 🕵')
    st.write('Source code for the project can be found at: https://github.com/SaadhJawwadh/SVD_AI')

# Page for predicting vulnerability
if choice == 'Predict Vulnerability':
    st.title('Sherlock AI 🕵')

    # slectbox for choosing model
    model_choice = st.selectbox('Select a model [Sherlock V.3 is the latest]', ['Sherlock V.3', 'Sherlock V.2', 'Sherlock V.1'])
    if model_choice == 'Sherlock V.3':
        model = tf.keras.models.load_model("models/Sherlock_V3.hdf5")
    elif model_choice == 'Sherlock V.2':
        model = tf.keras.models.load_model("models/Sherlock_V2.hdf5")
    elif model_choice == 'Sherlock V.1':
        model = tf.keras.models.load_model("models/Sherlock_V1.hdf5")

    # Text area for entering function
    st.write('Enter the function to be analyzed')
    function = st.text_area('Function:', '')
    function = function.lower()

    # Predict vulnerability on default model and handle errors
    if st.button('Predict'):
        if function == '':
            st.warning('Please enter a function to be analyzed')
        else:
        # Tokenize the function
            tokenized_function = tokenizer.texts_to_sequences([function])
            padded_function = tf.keras.preprocessing.sequence.pad_sequences(tokenized_function, maxlen=500, padding='post')

            # Pass the tokenized function to the model
            results = model.predict(padded_function)
            try:
                results = model.predict(padded_function)
            except Exception as e:
                st.error("An error occurred while predicting vulnerability: {}".format(e))
                st.stop()

            # Get the highest output value and its index
            max_value = np.max(results)
            max_index = np.where(results == max_value)
            max_index = max_index[0][0]

            # Display the vulnerability type in more readable format
            st.write('Vulnerability Type: ')
            if max_index == 0:
                st.write('There is significant probability (',max_value,') that the function has (CWE-119) Buffer overflow')
            elif max_index == 1:
                st.write('There is significant probability (',max_value,') that the function has (CWE-120) Improper Restriction of Operations within the Bounds of a Memory Buffer')
            elif max_index == 2:
                st.write('There is significant probability (',max_value,') that the function has (CWE-469) NULL Pointer Dereferences')
            elif max_index == 3:
                st.write('There is significant probability (',max_value,') that the function has (CWE-476) Use of Pointer Subtraction to Determine Size')
            elif max_index == 4:
                st.write('There is significant probability (',max_value,') that the function has Other Vulnerabilities (Improper Input Validation, Use of Uninitialized Variable, Buffer Access with Incorrect Length Value, etc.)')
            else:
                st.write('The function does not have any known vulnerabilities')

            # Display the probability for highest output value
            st.write('Probability: ', max_value)

```

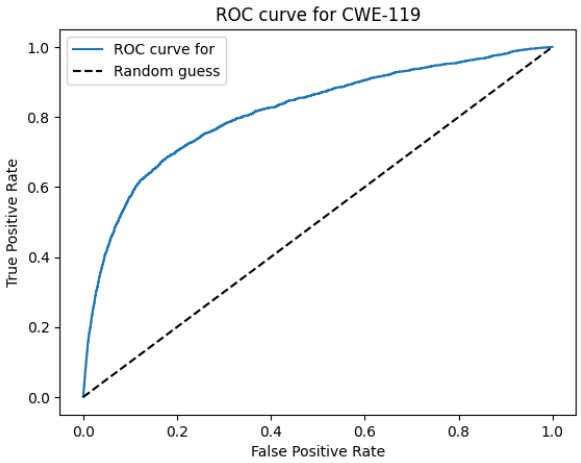
APPENDIX C – TEST RESULTS

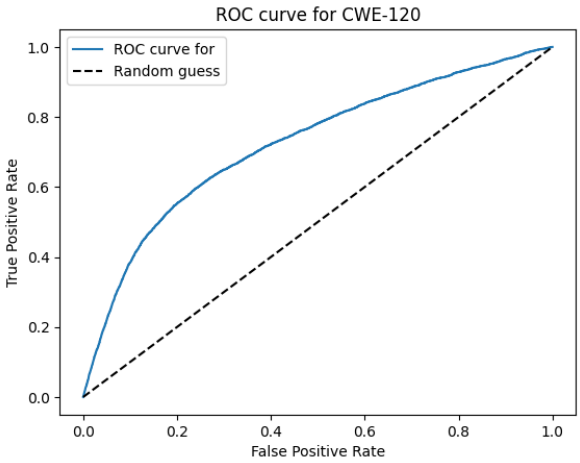


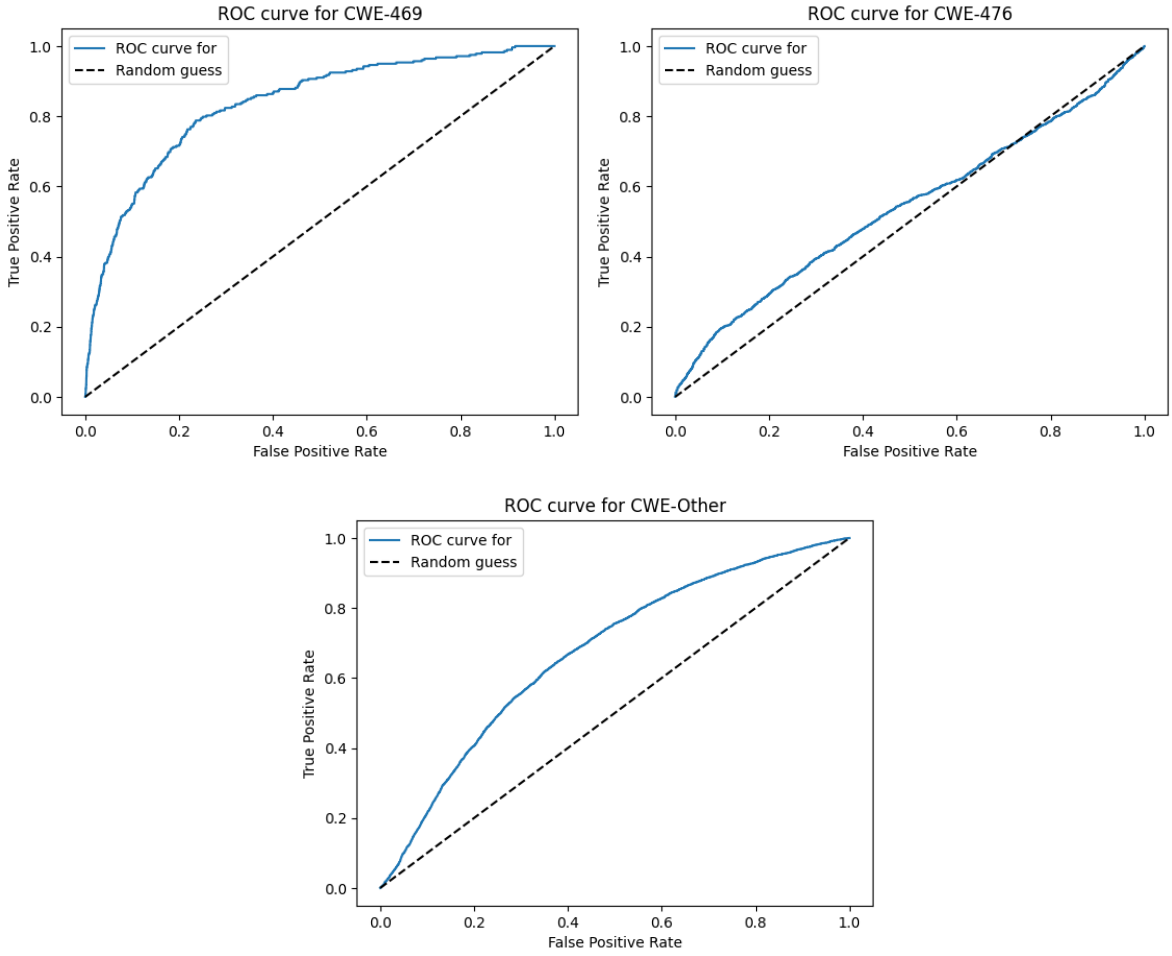
```
_____________________
Evaluation for CWE-119
_____________________

Accuracy: 0.9724844803365276
Precision: 0.2220464135021097
Recall: 0.17169657422512236
F1 score: 0.1936522539098436
Precision-Recall AUC: 0.12506375956458873
AUC: 0.8138499934292605
_____________________
_____________________
Classification Report for CWE-119
_____________________
              precision    recall  f1-score   support

           0       0.98      0.99      0.99    124967
           1       0.22      0.17      0.19      2452

    accuracy                           0.97    127419
   macro avg       0.60      0.58      0.59    127419
weighted avg       0.97      0.97      0.97    127419

_____________________
Evaluation for CWE-120
_____________________

Accuracy: 0.9233787739662059
Precision: 0.14929455801900374
Recall: 0.21202208137395215
F1 score: 0.17521331418433722
Precision-Recall AUC: 0.10613342272376702
AUC: 0.7239336310959966
_____________________
_____________________
Classification Report for CWE-120
_____________________
              precision    recall  f1-score   support

           0       0.97      0.95      0.96    122528
           1       0.15      0.21      0.18      4891

    accuracy                           0.92    127419
   macro avg       0.56      0.58      0.57    127419
weighted avg       0.94      0.92      0.93    127419
```

```
47  ------------------------
48  Evaluation for CWE-469
49  ------------------------
50
51  Accuracy: 0.997747588664171
52  Precision: 0.0
53  Recall: 0.0
54  F1 score: 0.0
55  Precision-Recall AUC: 0.017214625174684 14
56  AUC: 0.8358693591135068
57  ------------------------
58  ------------------------
59  Classification Report for CWE-469
60  ------------------------
61                precision    recall  f1-score   support
62
63             0       1.00      1.00      1.00    127141
64             1       0.00      0.00      0.00       278
65
66      accuracy                           1.00    127419
67     macro avg       0.50      0.50      0.50    127419
68  weighted avg       1.00      1.00      1.00    127419
69
70  ------------------------
71  Evaluation for CWE-476
72  ------------------------
73
74  Accuracy: 0.9827969141179886
75  Precision: 0.034450651769087 52
76  Recall: 0.031040268456375 84
77  F1 score: 0.032656663724624 89
78  Precision-Recall AUC: 0.014121206195202967
79  AUC: 0.5411645030634328
80  ------------------------
81  ------------------------
82  Classification Report for CWE-476
83  ------------------------
84                precision    recall  f1-score   support
85
86             0       0.99      0.99      0.99    126227
87             1       0.03      0.03      0.03      1192
88
89      accuracy                           0.98    127419
90     macro avg       0.51      0.51      0.51    127419
91  weighted avg       0.98      0.98      0.98    127419
92
```

```
92
93   ──────────────────────
94   Evaluation for CWE-Other
95   ──────────────────────
96
97   Accuracy: 0.9493874539903782
98   Precision: 0.04546850998463902
99   Recall: 0.042406876790830945
100  F1 score: 0.043884358784284656
101  Precision-Recall AUC: 0.04714639413408128
102  AUC: 0.6729844355145489
103  ──────────────────────
104  ──────────────────────
105  Classification Report for CWE-Other
106  ──────────────────────
107                precision    recall  f1-score   support
108
109             0       0.97      0.97      0.97    123929
110             1       0.05      0.04      0.04      3490
111
112      accuracy                           0.95    127419
113     macro avg       0.51      0.51      0.51    127419
114  weighted avg       0.95      0.95      0.95    127419
115
```






ROC curve for CWE-469
ROC curve for
Random guess
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve for CWE-476
ROC curve for
Random guess
True Positive Rate
False Positive Rate
ROC curve for CWE-Other
ROC curve for
Random guess
True Positive Rate
False Positive Rate

APPENDIX D – INTERVIWEE LIST

Interviewee	Category Id
Mr. Amal Wickramasinghe Technology Risk Manager at Axiata	CAT1
Mr. Nipuna Senanayake MS Computer Science (USA), Senior Lecturer -IIT	CAT2
Mr. Chamupathi Gigara Hettige Software Engineer at WSO2, B.Eng (Hons), Software Engineering	CAT1
Mr. Yasas Mahima PhD Candidate at UNSW Canberra, BEng (Hons) Software Engineering	CAT2
Anonymous Software Engineer Interna at Essva, BSc Computer science undergraduate	CAT3
Anonymous UI/UX Interna at TSG, BSc Computer science undergraduate	CAT3

APPENDIX D – USER INTERFACE