University of Mumbai

CyberDefend: Machine Learning Based Web Application for Cyber-Security

Submitted in partial fulfillment of requirements
For the degree of

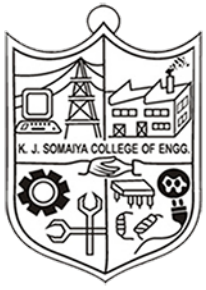
Bachelors in Technology

By

Gaurang Patil: 1814046
Gopalkrishna Waja: 1814062
Charmee Mehta: 1924003

Guide

Dr. Sonali Patil



Department of Information Technology
K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch 2018 - 2022

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Certificate

This is to certify that the dissertation report entitled **CyberDefend: Machine Learning Based Web Application for Cyber-Security** is a bonafide record of the dissertation work done by **Gaurang Patil**, **Gopalkrishna Waja**, **Charmee Mehta** bearing the **Roll no: 1814046, 1814062, and 1924003** in the year 2021-22 under the guidance of **Dr. Sonali Patil** of Department of Information Technology in partial fulfillment of requirement for the Bachelors in Technology degree in Information Technology of The University of Mumbai.

———————- ———————-

Guide Head of the Department

———————-

Principal

Date:

Place: Mumbai-77

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Certificate of Approval of Examiners

We certify that this report entitled **CyberDefend: Machine Learning Based Web Application for Cyber-Security** is a bonafide record of project work done by **Gaurang Patil**, **Gopalkrishna Waja**, and **Charmee Mehta**.

This project is approved for the award of Bachelors in Technology Degree in Information Technology of University of Mumbai.

_____

Internal Examiner

_____

External Examiner

Date:

Place: Mumbai-77

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

DECLARATION

We declare that this written report submission represents the work done based on our and/or others' ideas with adequately cited and referenced the original source. We also declare that we have adhered to all principles of intellectual property, academic honesty and integrity as we have not misinterpreted or fabricated or falsified any idea/data/fact/source/original work/ matter in our submission.

We understand that any violation of the above will be cause for disciplinary action by the college and may evoke the penal action from the sources which have not been properly cited or from whom proper permission is not sought.

_____- **Signature of the Student** **_____-** **Roll No.**	**_____-** **Signature of the Student** **_____-** **Roll No.**
_____- **Signature of the Student** **_____-** **Roll No.**	

Date:

Place: Mumbai-77

Dedicated to
our guide and colleagues

K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Abstract

The application of the machine learning (ML) technique in cyber-security is a rapidly expanding field that requires a significant deal of attention. The objective of this project is to provide an intelligent tool for the general public to help them recognize and prevent common cyber threats such as phishing and malware attacks that frequently accompany spam text messages. We provide an interface through which a user can detect dangerous IP addresses and spam messages(with or without URLs), discover past visits to malicious websites, and prevent visits to malicious/spam/phishing web pages.

Our project, titled "CyberDefend", is a comprehensive web application that serves as a platform to tackle the ever-increasing cyber crimes. Individuals looking for a smart tool to assist them in identifying and averting malicious cyber attacks can make use of our web application and leverage ML techniques for the same.

Keywords: **Cyber-security, Malicious, Safe Browsing, Spam, Phishing, Machine Learning**

Contents

List of Figures

List of Tables

Nomenclature

ACCC : Australian Competition Consumer Commission

Adaboost : Adaptive Boosting

ANN : Artificial Neural Network

API : Application Programming Interface

BOW : Bag Of Words

CAPTCHA : Completely Automated Public Turing test to tell Computers and Humans Apart

CNN : Convolutional Neural Network

CSS : Cascading Style Sheets

DMOZ : Directory Mozilla

DT : Decision Tree

HTML : Hypertext Markup Language

IP : Internet Protocol

JSON : JavaScript Object Notation

KL Divergence : Kullback-Leibler Divergence

KNN : K-Nearest Neighbours

LightGBM : Light Gradient Boosting Machine

ML : Machine Learning

NLP : Natural Language Processing

OS : Operating System

SMS : Short Message Service

SVM : Support Vector Machine

TF-IDF : Term frequency - Inverse Document Frequency

TLD : Top Level Domain

UCI : University of California Irvine

UI : User Interface

UML : Unified Modeling Language

UNB : University of New Brunswick

URL : Uniform Resource Locator

XGBoost : Extreme Gradient Boosting

CHAPTER 1

Introduction

This chapter gives a brief introduction to the thesis presented in the manner of the BTech Project undertaken. The problem definition is stated and is followed by motivation, the scope of the thesis, and salient contribution. Lastly, the organization of the thesis is written to help understand the entire document.

1.1 Problem Definition

The project objective is to build an intelligent tool for the general public that can aid them in detecting and preventing common cyberattacks by leveraging machine learning and deep learning techniques. These frequently occurring cybercrimes occur as a result of the distribution of spam messages and URLs leading to dangerous (phishing/malicious/spam) web pages, the URLs generally being a part of such spam messages. The tool specifically helps users to fight against cybercrimes such as phishing and malware attacks which often accompany spam messages and must achieve the following objectives-

- Leverage Machine learning / Deep learning techniques for
  - Spam message (with/without URL) detection including URL multiclass classification.
  - Discovering past visits to spam/malicious/phishing websites using uploaded browser history.

  - Safe Search to prevent visits to malicious/spam/phishing webpages.

- Detection of dangerous IP addresses and their involvement in malicious activity.

1.2 The motivation of the thesis

We were motivated to work towards developing a cyber-security web application based on the fact lately the cyber-crimes have been on a rise and as per an article published in The Hindu dated 9th March 2021, over 3.17 lakh cybercrimes have been reported in India in just 18 months. Additionally, according to Norton Labs, phishing attacks are the greatest threat to consumer cyber safety. Amidst the pandemic, cybercriminals have been using trending topics like vaccines, financial relief and tech support to trap victims into scams. SMS phishing has been on a rise.

As per ACCC, text messages are becoming the preferred way to trick people into scams. Though cyber security awareness is one of the key ways to resolve the issue, there is an urgent need to build a tool for the general public that can aid them in detecting and preventing common cyber attacks.

1.3 Scope of the thesis

The scope of this project is to develop a Web-based Cybersecurity application intended to help people having limited cyber-security awareness identify Phishing, Spam, and Malware dissemination attacks which are some of the most common yet dangerous cyber-attacks.

We have primarily focused on text messages and URLs as the medium

for carrying out such attacks. In addition to this, the platform also helps users to identify unauthorized involvement of their IP addresses in malicious activities.

1.4 Salient Contribution

The uniqueness of the project can be explained from the following two perspectives:

1. Research Perspective

   (a) The Multiclass classification of URLs would include the usage of domain-based and Javascript-based features apart from the lexical ones.

   (b) Trying resampling techniques to handle imbalanced data.

   (c) Inclusion of personally collected spam messages and usage of some advanced vectorization technique like Word2Vec or GloVe for spam message detection. Usage of CNN architecture for automated feature extraction.

2. Tool Perspective

   (a) Using blacklists in combination with ML models or neural networks for enhanced security. The tool would be more effective in case of zero-day attacks.

   (b) The tool would include both spam message detection and URL multiclass classification.

   (c) Facility to upload browser history and check if dangerous URLs have been visited previously.

1.5 Organization of the thesis

The synopsis of the report gives a brief description of each chapter in the project report.

1. Introduction: This chapter presents an introduction to the thesis at hand and discusses the problem definition while also giving a brief overview of how this project aims to solve the same. It also touches briefly on the topic of the scope of the thesis and its requirements.

2. Literature Survey: This chapter gives a brief description of why we chose this topic for our project. It also highlights why this project was needed and the work we did to set up a foundation before starting the project work, which includes referring to research papers and viewing competitive tools.

3. Software Project Management Plan: This chapter comprises the detailed project plan. Expected delivery dates of all documentation and project implementation have been mentioned.

4. Software Requirement Specification: This chapter mainly deals with the functional and non-functional requirements of the software and the necessary software system interfaces needed.

5. Software Design Document: This chapter deals with the software design and talks about the user interface in detail. The necessary software components, use cases, and UML diagrams are included here.

6. Implementation: This chapter provides an overview of the implementation details of the URL classifier, spam classifier, and web application.

7. Software Test Document: This chapter mainly deals with the testing approaches, test plans, and test cases that need to be run for effective testing of the software.

8. Results and Discussion: This section provides detailed results of the machine learning and deep learning models used to build the URL classifier and spam message detector. Discussion and explanations of the results have also been provided.

9. Conclusion and Future Work: This chapter describes the summation of arguments and conclusions drawn throughout the documentation as well as some ideas which can be implemented as part of future work.

10. References: This chapter contains the list of research papers and websites that were referred to create this project. The references are written in IEEE format.

11. Acknowledgement: This chapter expresses gratitude to everyone who has contributed to the success of this project through their assistance and support.

Summary: The project is mainly aimed at detecting and preventing common cyber-attacks like social engineering and malware. The massive increase in the number of such attacks in recent years motivates us to build this tool. The next chapter elaborates on existing literature which helps us build the tool.

CHAPTER 2

Literature Survey

This chapter provides a comprehensive literature survey used for the design and completion of the project. Papers from various reputed journals are referenced and cited here.

- **Towards Fighting Cybercrime: Malicious URL Attack Type Detection using Multiclass Classification [1]**
  - **Summary**

    The paper discusses multi-class URL classification using boosting Ensemble learners and works done in this paper can be used by us to implement features 2 and 3.
  - **Methodology**

    Here the authors have extracted 18 lexical and mathematical features like length, entropy, KL divergence, etc. from 1,26,983 URLs belonging to 4 classes Benign, Phishing, Malware, or Spam, and used these features to train XgBoost, AdaBoost, CatBoost, LightGBM models.
  - **Inference**

    They received the best result with AdaBoost which gave an overall accuracy of 0.95. Also, the detection accuracy for benign, phishing, and malware was greater than 0.96, but that of spam was comparatively low at around 0.85.
  - **Proposed Improvement**

Extract more discriminative features like domain-based and javascript-based features. Use imbalance reduction techniques like resampling to obtain better results.

- **Phishing Website Detection using Machine Learning Algorithms [2]**
  - **Summary**
    The paper specifically discusses the detection of phishing URLs using different ML algorithms and performs a comparative analysis to find the best model.
  - **Methodology**
    Here a combination of 16 lexical, domain and javascript-based features are extracted from 36,711 URLs and are used to train the Decision Tree Algorithm, Random Forest Algorithm, and Support Vector Machine Algorithm, and then their accuracy is compared for different split ratios.
  - **Inference**
    The random forest algorithm gave the best accuracy and lowest false-negative rate of 97.14 and 3.14 for a 90:10 split ratio. The authors have mentioned that in the future detection can be enhanced by using a blacklist in combination with this model.
  - **Proposed Improvement**
    Use the proposed features for multiclass classification. Use a blacklist in combination with Deep Neural nets for enhanced prediction.
- **SpaML: a Bimodal Ensemble Learning Spam Detector based on NLP Techniques [3]**
  - **Summary**
    The paper utilizes NLP in a combination of stacking ensemble

learning techniques for spam classification and the study was done here will be augmented by us to implement feature 1.

- **Methodology**

  Authors have used a dataset having 5574 messages (4827 non-spam 747 spam) and utilized BOW and TF-IDF for vectorization. After this classification is done using an ensemble learner using hard voting consisting of 7 models i.e. Multinomial NAive Bayes (MNB), Logistic regression (LR), SVM, nearest centroid center (NCC), Xgboost, KNN, and perceptron.

- **Inference**

  Received an overall accuracy of approx 97% and the results obtained by BOW and TF-IDF were comparable with BOW giving slightly better results, but the authors caution us that the results cannot be generalized unless the model is tested on some other datasets

- **Proposed Improvement**

  Use some advanced vectorization techniques like Word2Vec or GloVe. Train model on a Larger dataset with the inclusion of personally collected messages.

- **A Comprehensive Review on Email Spam Classification using Machine Learning Algorithms [4]**

  - **Summary**

    The paper discusses the binary classification of URLs using ML and makes use of statistical methods for feature selection. Again this paper can be used to form the basis for features 23

  - **Methodology**

    Here the Authors have used a dataset with 26,054 URLs (50% benign and 5% harmful) using which 41 features including Domain-

based and Alexa-based features were extracted and then statistical methods were used to select the top 17 features using which SVM, KNN, and gradient boosting algorithms were used.

  - **Inference**

    Using Boosting Ensemble gave a very high accuracy of 99% when tested using 10-fold cross-validation.

  - **Proposed Improvement**

    Use a larger dataset and perform multiclass- classification instead of binary classification. Also, observe the effect of giving a large number of features as an input to the deep neural net.

- **Intelligent Malicious URL Detection with Feature Analysis [5]**

  - **Summary**

    The paper provides a review of all the existing supervised, unsupervised and semi-supervised techniques for Spam classification. This paper helps us to select Supervised learning as the paradigm for spam classification.

  - **Methodology**

    This is a review paper where the authors have provided a Review of different algorithms under each category of learning technique and have. For, Supervised Algorithms, they have reviewed ANN, Naive Bayes, SVM, and DT while for Unsupervised Learning they have reviewed K-means.

  - **Inference**

    The high adoption rate for the supervised machine learning approach can be seen throughout the review. This approach is used mainly because it generates higher accuracy results with less variation giving high consistency.

  - **Proposed Improvement**

We, therefore, propose to use a Supervised Learning approach for Spam Classification.

- **Similar Tools Available Online**
  Here is a list of some similar and popular tools available online.
  - Google Safebrowsing
  - PhishTank
  - VirusTotal
  - APIvoid
  - urlscan.io
  - Norton Safeweb

  Some of these tools rely on blacklists and may not be very effective in case of zero-day attacks. The project's uniqueness from the perspective of existing tools has been provided in the "Salient Contribution" section in the introduction of the synopsis.

Summary: We can use advanced vectorization techniques like Word2Vec and GloVe for spam classification and need to use CNN architecture for automated feature extraction. We can classify URLs into 4 categories namely- safe, spam, phishing and malware using lexical features and HTML-JS based features. The next chapter elaborates on the project management plan.

CHAPTER 3

Software Project Management Plan

This chapter illustrates the Software Project Management Plan document. Content for project deliverables, project organization, roles and responsibilities of the team members, tools and techniques to be used, and finally the essential tasks and phases of the project; is presented here. The Gantt chart for the project schedule is attached to this chapter.

3.1 Introduction

3.1.1 Project Overview

CyberDefend is a web application for the general public which can aid them in detecting and preventing common cyberattacks. It mainly consists of different integral features such as detection of spam messages (with or without URLs), discovering past visits to malicious websites, safe search, and detecting dangerous IP addresses. The application takes input data from the user in the form of text, URLs, JSON, and IP addresses and classifies the data as malicious or benign. It's a software program, designed to tackle the rising challenges in cybersecurity faced by the netizens.

The purpose of this document is to present a detailed description of the ML Based Web Application for Cybersecurity Project. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, and the constraints under which it must operate. This

document is intended for both the users and the developers of the system. The main purpose of the software is to provide the citizens with a way to overcome cybercrimes and simplify cybersecurity.

The application works on web-based platforms on various OS and is to be created using Streamlit Framework, Python, and a few supporting technologies. It can be used by a broad spectrum of people in various sects and arenas of life with respect to their educational level, experience, and technical expertise. SPMP document is created to list the intended audience and provides suggestions for the same. This document is mainly for both developers and project managers to determine the procedure in the creation of further phases of the application and determine a road map or workflow for the same.

This document is mainly for both developers and project managers to determine the procedure in the creation of further phases of the application and determine a road map or workflow for the same

3.1.2 Project Deliverables

Project deliverables are the results of the project or the processes involved in the project. For our project, the deliverables are listed in table 3.1 located below.

Project Deliverables	Description	Delivery Date
Software Project Management Plan (SPMP)	It will include the outlines of the project and people involved and details process and guidelines to be followed right through the project	01-10-2021
Software Requirements Specifications (SRS)	It will include details about the functional and non-functional requirements necessary for the project	01-10-2021
Software Design Document (SDD)	Contains an in-depth design of the solution, including usecase, collaboration, sequence and class diagrams	15-10-2021
Software Test Document (STD)	Contains an in-depth collection of test cases that are tested on the solution and compared with the expected results	15-10-2021
Synopsis	It will include the overview of the project with the design of the application, execution plan and test cases	05-11-2021
Black Book	Details how the system runs and instructs the user on how to use the application properly	30-04-2022
Project Submission	The complete application with all the features	30-04-2022

Table 3.1: Project Deliverables

3.2 Project Organization

3.2.1 Software Process Model

The requirements of the problem are well understood. The workflows from communication through deployment in a reasonably linear fashion. The waterfall model, as shown in Fig 3.1, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in the ongoing support of the completed software.

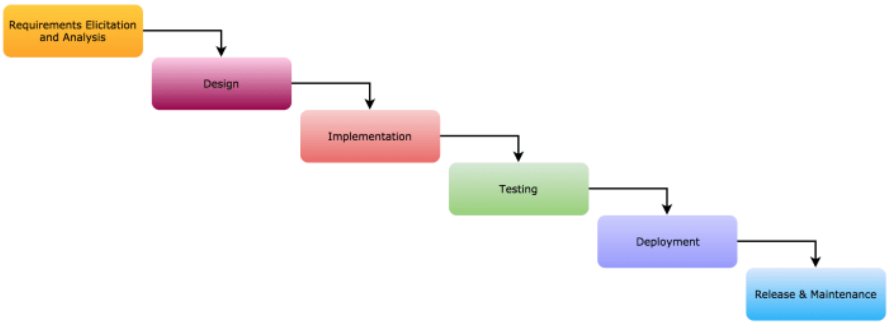


Figure 3.1: Waterfall Model

The following are also some of the reasons for using this model for the project:

1. The web application development phase of the project is small and the requirements are well defined.
2. Management becomes very convenient: Each phase has well-defined outputs and review processes associated with itself, which helps in managing the project well.
3. Output of each phase is clearly defined in the software documentation.

For building the machine learning/deep learning models, an iterative approach needs to be adopted so that the performance metrics of the models can be improved. The iterative approach will allow the developers to build better models and will grant them enough freedom to try out new approaches. Based on the observations during the implementation of the project, the developers can modify their approach as per the need.

3.2.2 Roles and Responsibilities

Considering the withdrawal of one of the team members, the corresponding roles and responsibilities shown in table 3.2 will be divided among the rest of the members for semester 8.

Roles	Responsibilities
Project Manager - Gopalkrishna Waja	The responsibility of the project manager is to assign roles and responsibilities to team members and will look after all the aspects of the project
Designer - Esha Vats, Charmee Mehta	The responsibility of the designer is to create a design for the project which will help to gain insights for a better understanding of the system
Developer - Gaurang Patil, Gopalkrishna Waja, Esha Vats, Charmee Mehta	The responsibility of the developer is to understand the design and convert it into a feasible working solution
Tester - Esha Vats, Charmee Mehta	The responsibility of the tester is to create test cases and evaluate the solution against the requirements that have been gathered in previous phases of the development life cycle
Analyst - Gaurang Patil	The responsibility of the analyst is to manage the development of the project through special research, data analysis, and data collection to facilitate strategic decision-making

Table 3.2: Roles and Responsibilities

3.2.3 Tools and Techniques

1. **UML diagram tool**: StarUML, draw.io, Lucidchart
2. **Wireframing**: Adobe XD, Figma
3. **Documentation**: LATEX using Overleaf
4. **Development**: Streamlit, HTML, CSS, Javascript, Bootstrap, Python, Jupyter Notebook, Neutrino API, Google Safe Browsing Lookup API, Cisco top 1M domains CSV file
5. **Gantt Chart**: GanttProject
6. **Testing tools**: SauceLabs, GTmetrix

3.3 Project Management Plan

3.3.1 Requirements Gathering and Analysis

1. **Description**:
   Requirement analysis involves tasks that go into determining the needs to be met for a software project. It takes into account conflicting requirements, analyzing, documenting, validating, and managing software requirements.

2. **Deliverable and Milestones**:
   Documentation of requirement gathering, Understanding of business requirements and software requirements.

3. **Resources Needed**:
   Frequent access to customer interactions to understand the requirements of the customer, Access to stationery such as paper, pen, and computer.

4. **Dependencies and Constraints**:
   Success criteria are not defined clearly, Customers change their minds, Customers are not willing to speak up or they are being too expressive, Clients imply or insist on a particular technical solution, Clients have Conflicting priorities

5. **Risks and Contingencies**:
   Unclear understanding of the problem. This can be avoided by frequent meetings with the client and by choosing a relevant process model

3.3.2 Design

1. **Description**:
   In this task, we focus on developing the UI Designing which plays

an important part in how the user will be interacting with the product software, and also designing the UML diagrams which define how the product will be created to capture the system's functionality and requirements along with its dynamic behavior.

2. **Deliverable and Milestones**:
   Upon completion of this task, upon discussion with the stakeholders and getting the opinions of the users, a specific UI design appropriate for the product will be finalized and developed. Upon discussion with the team members and the designer, the UML Diagrams will be finalized and developed, which serves as the backbone of developing the product.

3. **Resources Needed**:
   For designing the of UML Diagrams and Wireframing tools such as Adobe XD and Figma, StarUML, creatly and draw.io are required for the same.

4. **Dependencies and Constraints**:
   The design approach that is going to be used for the product should be suitable for the product architecture.

5. **Risks and Contingencies**:
   Risk would be miscommunication between the product development team as that would lead to the ill design in the UML Diagrams which is the backbone of the development cycle of the entire product software.

3.3.3 Implementation and Coding

1. **Description**:
   Coding is the third phase of software development. Detailed design specifications are used as the input by developers to build the software product. The main focus of this phase is development. The entire design will be broken into modules and developers will work on individual modules, then they will integrate the separate modules into one system finally. The software will be made in 4 phases : Detection of Spam messages Module, Safe Search Module, Detection of Dangerous IP Addresses Module, and Discovering past visits to malicious websites Module.

2. **Deliverable and Milestones**:
   The final product after this phase is the software that is an intelligent tool for the general public that can aid them in detecting and preventing common cyber attacks.

3. **Resources Needed**:
   For coding and development purposes, we would need Streamlit, HTML, CSS, Javascript, Bootstrap, Python, Jupyter Notebook, and External APIs like Neutrino.

4. **Dependencies and Constraints**:
   The solution is dependent on the design which should be feasible and appropriate.

5. **Risks and Contingencies**:
   The risk would be miscommunication between the product development team as that would lead to the ill design in the UML Diagrams which is the backbone of the development cycle of the entire product software.

3.3.4 Testing

1. **Description**:
   We test the product by creating test cases for the product and the product is tested against those test cases. The model will be tested on data that it has never seen to authenticate the accuracy of the model.

2. **Deliverable and Milestones**:
   The creation of test cases of the product and testing of the product to validate the test cases. The model will be tested on unseen data to get an accurate output of the model.

3. **Resources Needed**:
   The tools required for testing the model are Jupyter Notebook & matplotlib for inferencing the model for unseen data and using Sauce-Labs for compatibility testing of the web application. We use GT-metrix for usability testing.

4. **Dependencies and Constraints**:
   As many test cases should be generated as we can, also product should pass more test cases. Test data should be of appropriate dimension.

5. **Risks and Contingencies**:
   If the product is tested against wrong test cases or incomplete test cases then the product might not work properly. Inappropriate dimensions may lead to abrupt results.

3.3.5 Deployment and Improvisation

1. **Description**:
   The solution is being deployed on a platform that will make it con-

venient to use and visualize the solution.

2. **Deliverable and Milestones**:
   Deployed software and Customer review along with a Live Production environment act as a deliverable here. Updated Software and bug fixes may be expected as deliverable as well.

3. **Resources Needed**:
   Tools needed for Deployment and Maintenance include Deployed Software and Regression Testing Tools.

4. **Dependencies and Constraints**
   Meetings with the stakeholders and users so as to determine the finality of the product software. Customer review is needed to complete this phase.

5. **Risks and Contingencies**:
   Risks would be a failure to get clarity on the challenges from hidden bugs and real-world unidentified problems and discrepancies along with miscommunication between the users and the developers.

3.3.6 Assignments

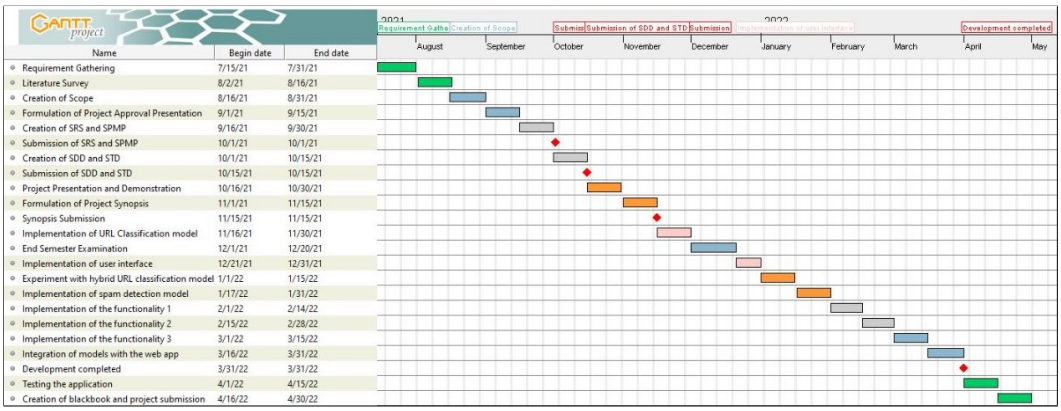
Considering the withdrawal of one of the team members, the corresponding role shown in table 3.3 will be taken over by the rest of the members for semester 8.

Name	Roles
Gopalkrishna Waja	Project Manager/Developer
Gaurang Patil	Analyst/Developer
Esha Vats	Designer/Developer/Tester
Charmee Mehta	Designer/Developer/Tester

Table 3.3: Assignments

3.3.7 Timetable

Fig 3.2 shows the Gantt Chart for the project



Name	Begin date	End date
Requirement Gathering	7/15/21	7/31/21
Literature Survey	8/2/21	8/16/21
Creation of Scope	8/16/21	8/31/21
Formulation of Project Approval Presentation	9/1/21	9/15/21
Creation of SRS and SPMP	9/16/21	9/30/21
Submission of SRS and SPMP	10/1/21	10/1/21
Creation of SDD and STD	10/1/21	10/15/21
Submission of SDD and STD	10/15/21	10/15/21
Project Presentation and Demonstration	10/16/21	10/30/21
Formulation of Project Synopsis	11/1/21	11/15/21
Synopsis Submission	11/15/21	11/15/21
Implementation of URL Classification model	11/16/21	11/30/21
End Semester Examination	12/1/21	12/20/21
Implementation of user interface	12/21/21	12/31/21
Experiment with hybrid URL classification model	1/1/22	1/15/22
Implementation of spam detection model	1/17/22	1/31/22
Implementation of the functionality 1	2/1/22	2/14/22
Implementation of the functionality 2	2/15/22	2/28/22
Implementation of the functionality 3	3/1/22	3/15/22
Integration of models with the web app	3/16/22	3/31/22
Development completed	3/31/22	3/31/22
Testing the application	4/1/22	4/15/22
Creation of blackbook and project submission	4/16/22	4/30/22

Figure 3.2: Gantt Chart

Summary: The waterfall process model will be followed for the development of the web application whereas an iterative approach will be followed for building the machine learning models. Streamlit would be used for developing the web application and Jupyter Notebook would be used for training and testing of the models. The next chapter elaborates on the specific requirements of the software and the functional and non-functional requirements of the software.

CHAPTER 4

Software Requirement Specification

This chapter provides the Software Requirements Specifications document; and explains the requirements of the project. It includes all functional requirements ranging from, and inclusive the inputs to the system to the output and its analysis. Lastly, the non-functional requirements enlist all relevant performance criteria.

4.1 Introduction

This document is intended to provide the software specifications and requirements of our cyber-security web application which will serve as a road map for developing upcoming phases like the design and implementation of the web application. This document will provide a detailed description of the parameters and goals of our web application. This is done by first providing a product overview and the target audience of the product, followed by a comprehensive elucidation of the specific requirements and software system attributes including user interface, hardware, and software requirements. This document is prepared for the use of both the developers and the owner of the product. Due to the withdrawal of one of the group members, the functionality of solving a CAPTCHA before form submission will not be implemented as a part of the project. The functionality of the 'Dangerous IP address detector' will not be implemented for the same reason. Corresponding technologies needed to build the same have been mentioned in the document, but will not be

used to build the web application.

4.1.1 Product Overview

CyberDefend is a cyber-security web application that leverages concepts of machine learning and deep learning to safeguard users from common yet pernicious cyber attacks like spam, phishing, malware dissemination, and zero-day attacks. The web application's primary audience is innocent users who have limited cyber-security awareness who can use CyberDefend to verify the whether a URL or a message is safe, identify past visits to malicious websites, and get information about if the user's device is involved in a malicious activity like distribution of malicious content.

4.2 Specific Requirements

This section of the SRS contains all of the requirements required to allow the developer to understand and check the original system requirements. It also helps the designer to design a system to satisfy those requirements, and the tester to test that the system satisfies those requirements. The requirements stated in this section are intended to be correct, concise, consistent, unambiguous, traceable, and verifiable.

4.2.1 External Interface Requirements

4.2.1.1 User Interfaces

The user interfaces for the CyberDefend web application are shown in fig 4.1 to 4.4.

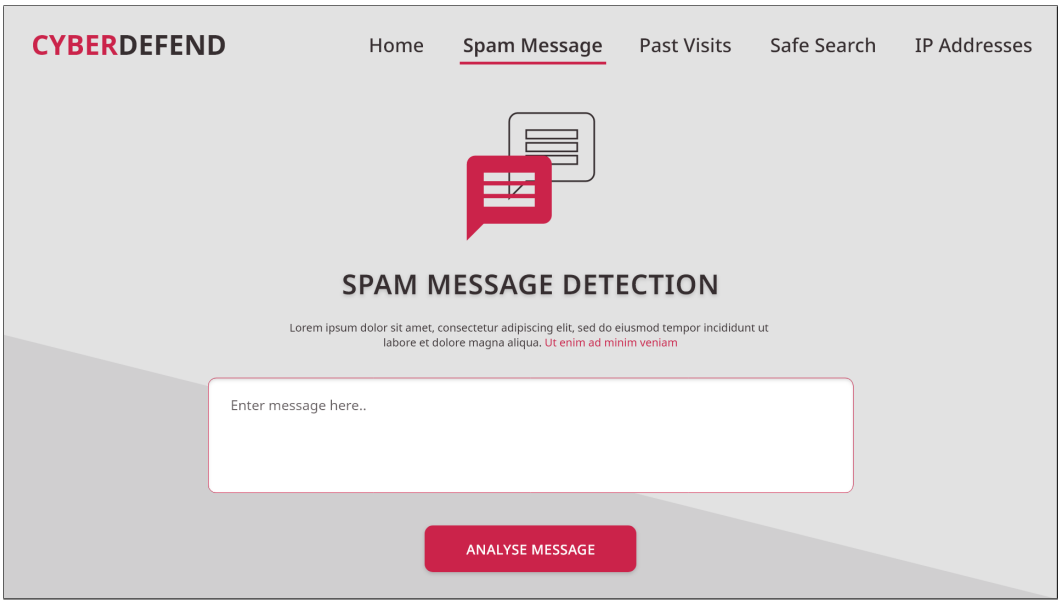


Figure 4.1: Spam Messages: Initial

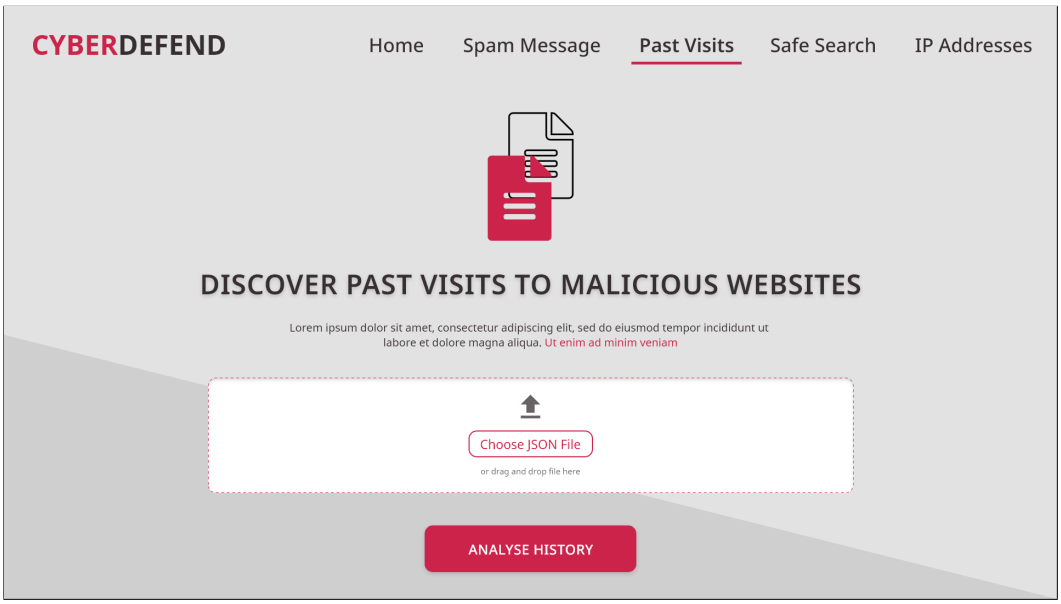


Figure 4.2: Past Visit: Initial

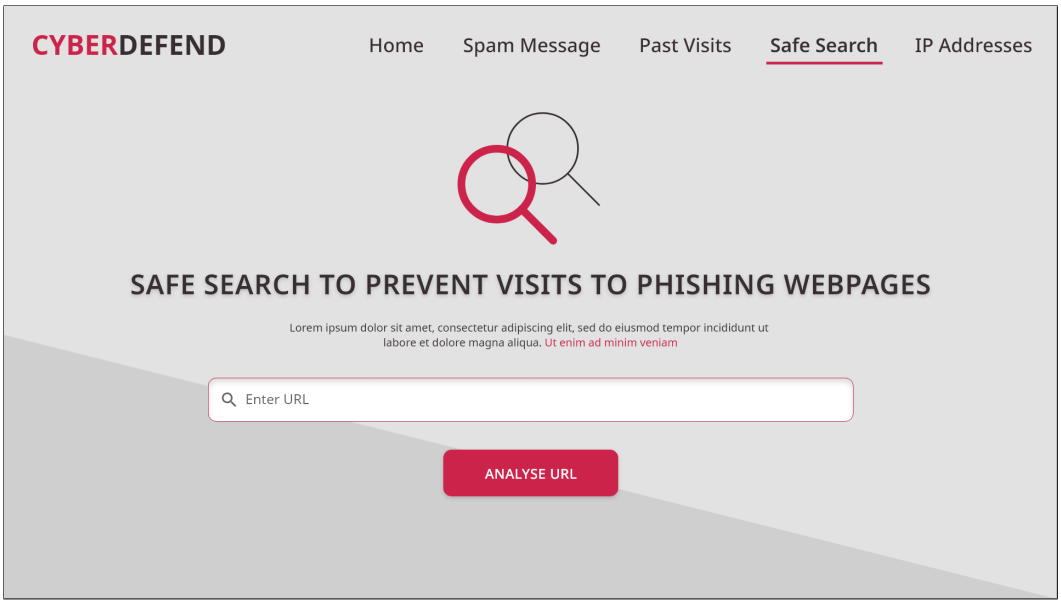


Figure 4.3: Safe Search: Initial

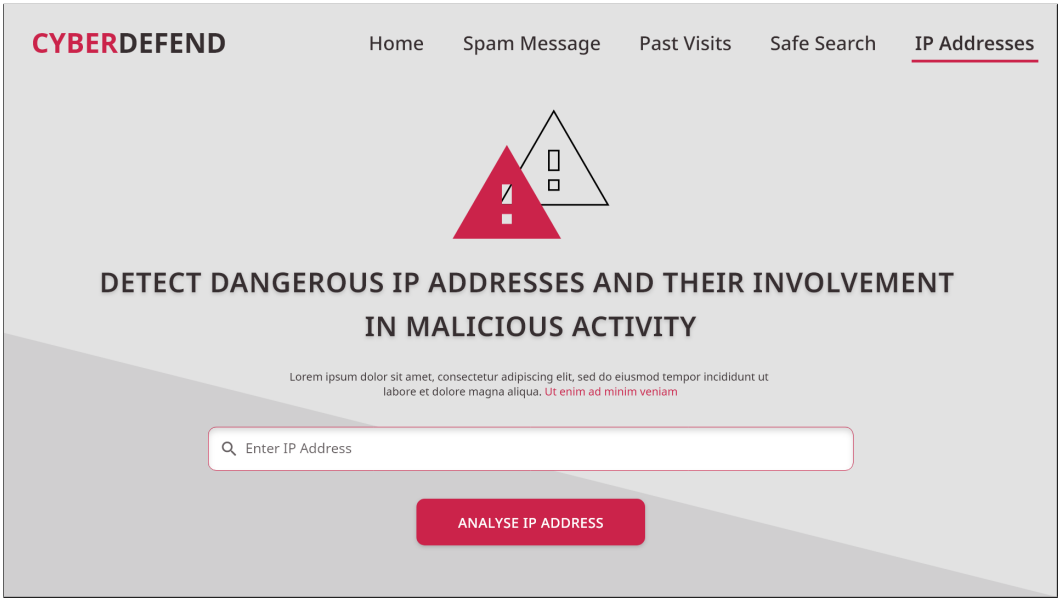


Figure 4.4: IP address: Initial

4.2.1.2 Software Interfaces

CyberDefend web application will require to interface with the following software, packages, and systems for optimal functioning. As the machine learning functionalities are to be developed in an iterative approach, the table mentions a few important libraries which would be used for development. The complete list of libraries and their versions can be found in the 'requirements.txt' file.

Name	Version	Type	Purpose
WhoIs	0.7.3	API / Library	Query Domain-based URL Features
Neutrino		API	Verify Involvement of IP in Malicious Activity
TensorFlow	2.5.0	Library	Creation of Deep Learning models
Streamlit Cloud	-	Software	Hosting the Web application
Google SafeBrowsing Lookup		API	To check if URL is blacklisted or not.

Table 4.1: Software Interfaces

4.2.1.3 Communications Protocols

The communication using Neutrino API will be in a "RESTful" manner and it will accept request parameters as either HTTP GET or HTTP POST. When sending API requests as HTTP POST the content will be accepted in JSON format. Along with the HTTP request, authentication parameters must also be sent in form of API key user ID. The response to this request will be in form of a JSON file from which require content will be extracted. Other than this the python whois package will send a query to the WhoiS server to obtain Top-Level-Domain (TLD) informa-

tion about the queried URL which is stored in form of a registry, which maintains the centralized directory for each TLD.

4.2.2 Functional Requirements

1. **Spam message Detection**

   This functionality safeguards the user from dangerous text messages by helping the user to identify whether a text message is "Spam" or "Not Spam" and classifies the URLs in the message, if any, as "Safe", "Phishing", "Spam" or "Malware". The following are the specific specifications of this feature:

   - The User must paste the text message, whose safety needs to be verified, in the text box provided on the web application.
   - The text message may or may not have URLs.
   - Images, Video, and other media items cannot be copied to the text box.
   - This functionality must provide a button named "Verify/Classify" which when clicked classifies the text message as "Spam" or "Not Spam" and the URL, if present, as "Safe", "Phishing", "Spam" or "Malware".
   - If URL is present in the text the result must be shown in a tabulate manner with columns, "Site name, URL, Category" where the category is either "Phishing", "Spam" or "Malware".

2. **Past malicious websites discovery**

   This Functionality allows user to discover their visits to possibly malicious websites. Here, the system, upon submission of browser history by the user in JSON format, will ferret the malicious websites and display the results in a tabular format. The following are the specific specifications of this feature:

- The web application must provide a file upload button to accept user history in JSON format only.
- The web application must provide a check history button, which when clicked after uploading the JSON history file runs the script to find malicious visits. Clicking the check button before uploading the file gives a message "Please upload your browsing history in JSON format first".
- If the application detects that the user has visited some possible malicious websites then the system must display all such visits in a tabular format with columns, "Site name, URL, Category" where the category is either "Phishing", "Spam" or "Malware".
- If no visits to malicious websites are detected then the system must display "Your browsing activity seems to be safe".

3. **Safe Search**

   Safe Search aids the user to identify malicious and suspicious URLs and hence safeguards users from accidentally landing on malicious websites. The following are the specific specifications of this feature:

   - The application must provide a text box in which the user will type or paste the URL he wants to visit.
   - When the user hits "Enter" upon entering the URL the system should classify the URL as "Safe", "Phishing", "Spam", or "Malware".
   - If the URL is classified safe then the user must be provided with an option to be redirected to the corresponding URL
   - If the URL is "Phishing", "Spam" or "Malware" the user must be given a warning message indicating the class of the URL.

4. **Detection of dangerous IP addresses**

   This functionality aids the user in knowing if a given IP address has been blacklisted or not. If yes, it can also provide details of the cybercriminal activity associated with the IP address. The following are the specific specifications of this feature:

   - The application must provide a text box in which the user will type the IP address and submit it.
   - When the IP address is submitted, the result must display whether the IP address is present on the blacklist or not. If present, the details of cybercriminal activity must be displayed, if available.

4.2.3 Non-Functional Requirements

4.2.3.1 Availability

CyberDefend must provide all-around availability so that users can utilize the functionalities as per their needs in different time zones.

4.2.3.2 Security

Security requirement ensures the safety of the web application against malicious activity aimed at sabotaging or disrupting the system. Thus the following security requirements must be met to ensure the safety of the system:-

- Input sanitization must be done at all places where the user gives input to prevent the injection of harmful code into the system.
- User must solve a CAPTCHA before submitting a form. This is to prevent automated bots from submitting requests to the server. It also ensures that denial of service attacks are not launched by automated bots.

4.2.3.3 Portability

Portability defines the usability of the software in various environments like web browsers or operating systems. For CyberDefend the following are the portability requirements:-

- CyberDefend must be accessible using any web browser over the internet.
- CyberDefend must work on all devices- desktops, laptops, and smartphones, irrespective of their operating systems.

4.2.3.4 Maintainability

Maintainability can be understood as the time required for a component to be fixed. To ensure that the web application is maintainable the following requirement should be fulfilled:-

- All predictions using machine learning or deep learning models must be facilitated through a function call. This ensures that when updated models are deployed in the future, changes in the code are minimal.

4.2.3.5 Usability

Usability essentially refers to how easy it is for the customers or the users to use the system. The following are the usability requirements:-

- The user interface must be simple to use and must be self-explanatory. Anyone with basic computer literacy must be able to use CyberDefend. For the feature involving the discovery of past visits to dangerous websites, the user must know how the browser history can be downloaded as a JSON file using his/her Google account.

- The developers should provide a user manual to guide the users about the functioning of each functionality.

Summary: The web application needs to be capable of detecting spam messages, malicious URLs and discovering past visits to such dangerous websites. The application should be deployed on cloud so as to ensure all round availability. The next chapter elaborates on the design details of the web application.

CHAPTER 5

Software Design Description

This chapter includes the Software Design Description document. Content for the design overview, system architectural design, and detailed description of the components of the web application with UI design wire-frames are presented here.

5.1 Introduction

The SDD provides design information related to the planning, analysis, and implementation of CyberDefend. The software system is partitioned into multiple components whose descriptions and interaction details are provided in SDD.

5.1.1 Design Overview

A client-server architecture is suitable for CyberDefend. In a client-server architecture, the server is capable of serving multiple client requests simultaneously and maintenance becomes easy. New/updated machine learning or deep learning models can be integrated into the web application on the server-side with ease. Design components of CyberDefend include a spam message detector, dangerous URL classifier, URL feature extractor, and dangerous IP address detector.

5.1.2 Requirements Traceability Matrix

The matrix given below shows how each feature mentioned in the SRS is supported by the design components of the software system.

	Spam message detector	Dangerous URL classifier	URL feature extractor	Dangerous IP address detector
Spam Message Detection	X	X	X	
Past malicious websites discovery		X	X	
Safe Search		X	X	
Dangerous IP address detector				X

5.2 System Architectural Design

5.2.1 Chosen System Architecture

Client – Server Architecture

The client-server architecture is a distributed structure wherein the remote server acts as a provider of a resource or a service to the local clients who send requests to the server. The server is capable of providing service to multiple clients. The client sends a request to the server

over the internet and the server responds to the client's request.

Advantages:

- Backups can be taken very easily due to centralization.
- Maintenance is easy as upgrading and repairing are required only at the server-side.

Disadvantages:

- The server may be overloaded with requests from the client and the congestion may affect the latency of the web application. This makes such an architecture prone to denial of service attacks. To counter this, all users need to solve a CAPTCHA before submitting any form. This significantly reduces the chances of a denial of service attack.
- If the server is down, the client's requests cannot be attended to, which means that the architecture is not very robust.

5.2.2 Discussion of Alternative Designs

The alternative to using a client-server architecture will be using a Microservice-based architecture. This is possible as our web application mainly comprises 4 different functionalities each of which can be viewed as an independent microservice. Since microservices make use of modularizing software changes and maintenance can be done much faster and easier when compared to traditional monolithic architectures. Microservices are also extremely helpful to improve scalability and often have very little fault detection and maintenance time. Although microservices have these benefits the initial cost and complexity of setting up microservices are high and since this application is developed as part of a project and will be used in a small environment client-server monolithic architecture

makes more sense than using microservices but in the future, if this application is to be scaled up we can go on to adopt the microservice-based architecture.

5.2.3 System Interface Description

5.2.3.1 Software Interfaces

CyberDefend would be using WHOIS API/library to extract the domain-based features of the URL being analyzed. Google SafeBrowsing Lookup API would be used to check blacklisted URLs. Neutrino API would be used to check the given IP address against a blacklist. TensorFlow would be used to create deep learning models. Streamlit cloud would be needed to host the web application.

5.3 Detailed Description Of Components

5.3.1 Spam Message Detector

This component uses a deep learning model to classify a given message as spam or ham. The deep learning model needs to be trained and tested on the available data before integrating it with the web application. Only after the trained model is integrated with the web application, the component is functional and ready to use.

5.3.2 Dangerous URL Classifier

This component uses machine learning or a deep learning model to classify a given URL as benign/spam/phishing/malicious. Various machine learning models need to be trained and tested on the available data to find a suitable model which can be integrated into the application. Only after

the trained model is integrated with the web application, the component is functional and ready to use.

5.3.3 Dangerous IP Address Detector

This component uses the Neutrino API to compare a given IP address against a list of blacklisted IP addresses. Blacklisted IP addresses can be detected by this component. The type of malicious activity the IP address is involved in can also be detected provided the API response includes these details.

5.3.4 URL Feature Extractor

This component extracts the lexical, Javascript, and domain-based features of the URL which serve as input to the dangerous URL classifier. The component needs the URL as input from the user. The component also relies on WHOIS queries to extract the domain-based features.

5.3.5 Past Visits to Dangerous Sites Detector

This component uses the URL feature extractor and dangerous URL classifier to classify the URLs present in the user's uploaded browser history as safe/spam/malicious/phishing and displays the results in a tabular format.

5.4 User Interface Design

5.4.1 Description of User Interface

5.4.1.1 Screen Images

Figure 5.1: Home Page 1

Screen Description: fig 5.1 has the main menu on the top consisting of five choices. Users can jump to the respective page as per the functionality mentioned in the menu. If the user clicks on the choice "Spam Message", the user is directed to the page where the user can enter the spam message which needs to be considered for analysis. If the user clicks on the choice "Past Visits", the user is directed to the page where the user can upload his/her browser history for analysis. If the user clicks on the choice "Safe Search", the user is directed to the page where the user can enter the URL which needs to be analyzed. If the user clicks on the choice "IP addresses", the user is directed to the page where the user can enter the IP address which needs to be analyzed. The user also gets an option to jump to the home page by clicking on "Home" or by clicking on the logo of CyberDefend.

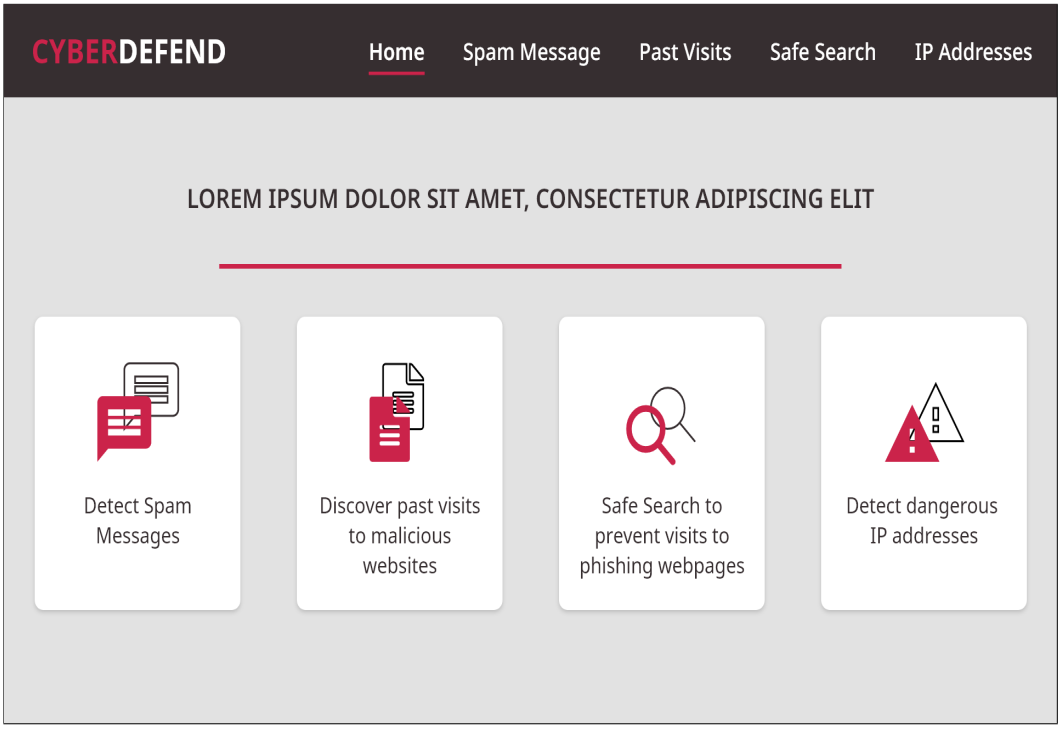


Figure 5.2: Home Page 2

Screen Description: In fig 5.2 when the user scrolls down the home page, the four main functionalities of CyberDefend are made visible.

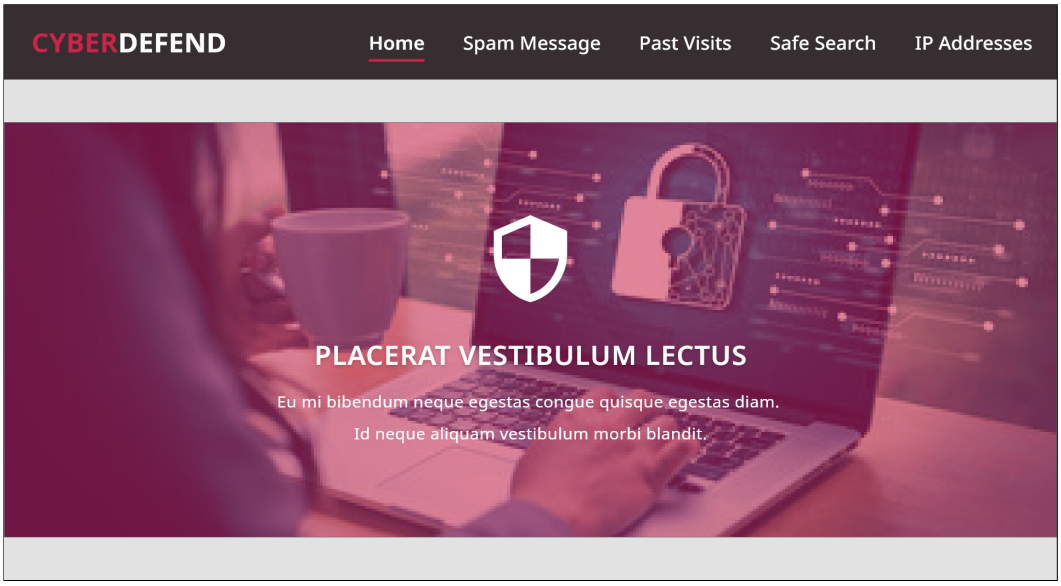


Figure 5.3: Home Page 3

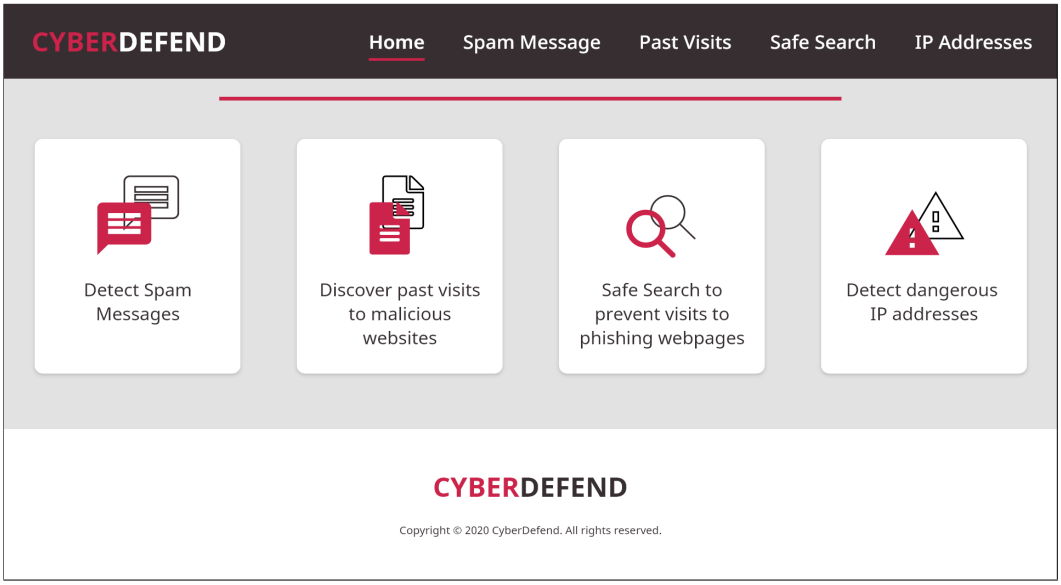


Figure 5.4: Home Page 4

Screen Description: fig 5.4 shows footer is placed at the bottom of the page along with copyright information.

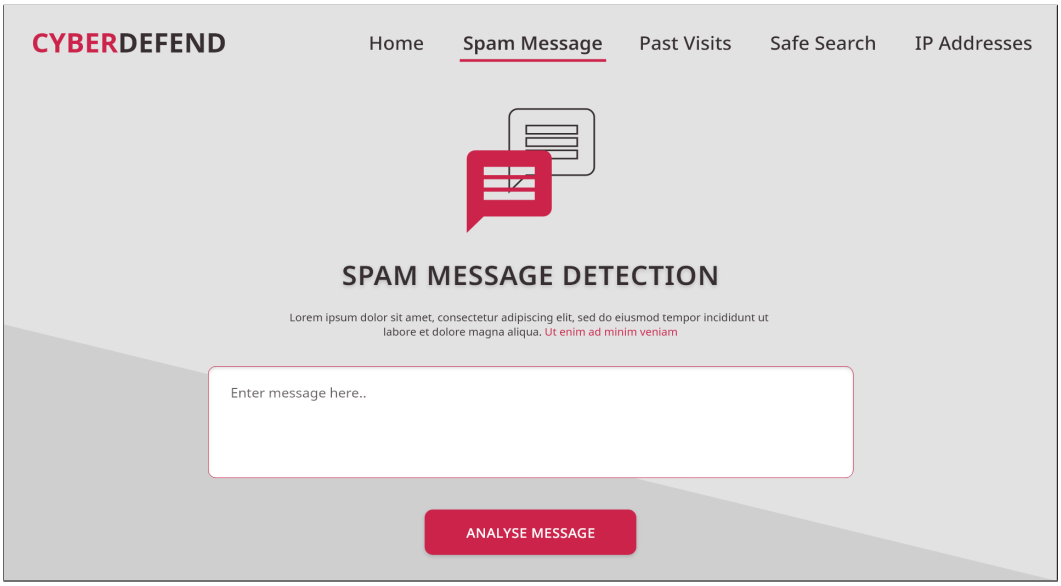


Figure 5.5: Spam Messages: Initial

Screen Description: fig 5.5 shows the user is provided with a text box where the message which needs to be analyzed can be pasted. When the user clicks on the "Analyse Message" button, the message is sent for analysis on the server-side.

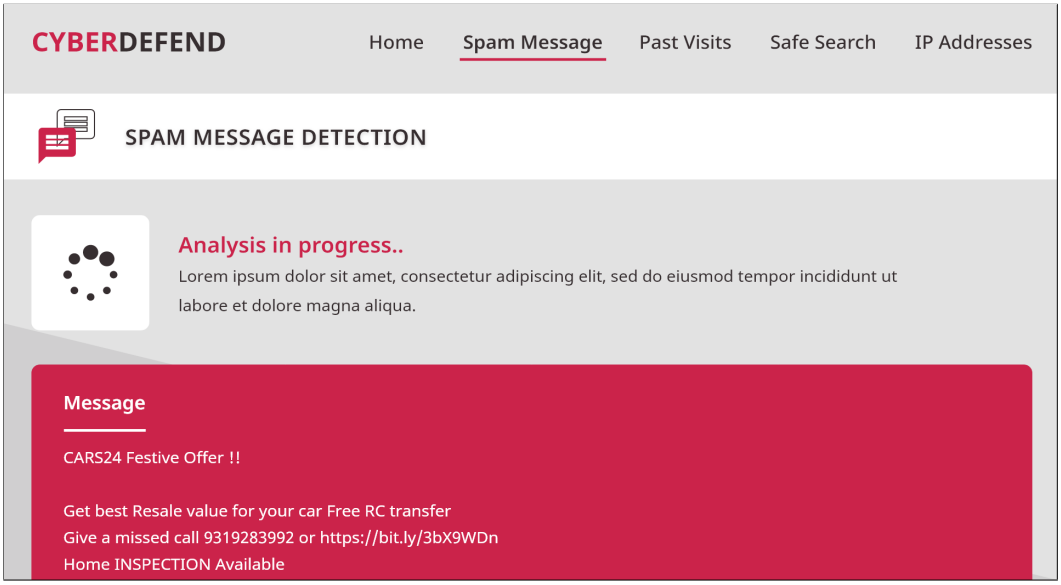


Figure 5.6: Spam Messages: Analysis

Screen Description: fig 5.6 shows as soon as the message is sent for analysis, the user is informed that the analysis is in progress.

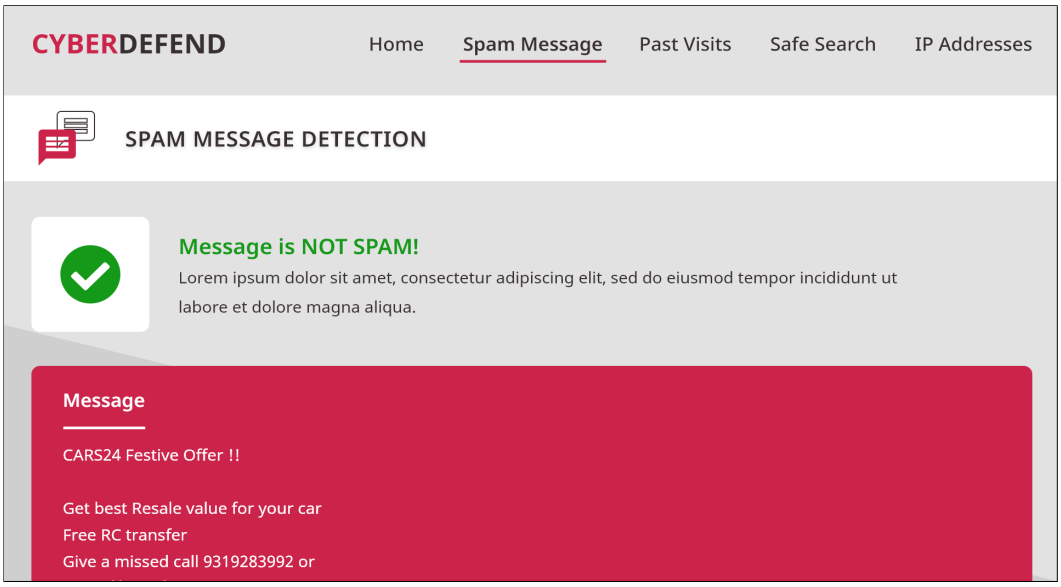


Figure 5.7: Spam Messages: Not Spam

Screen Description: fig 5.7 shows if the message is not spam, a message is displayed saying that it is not spam.

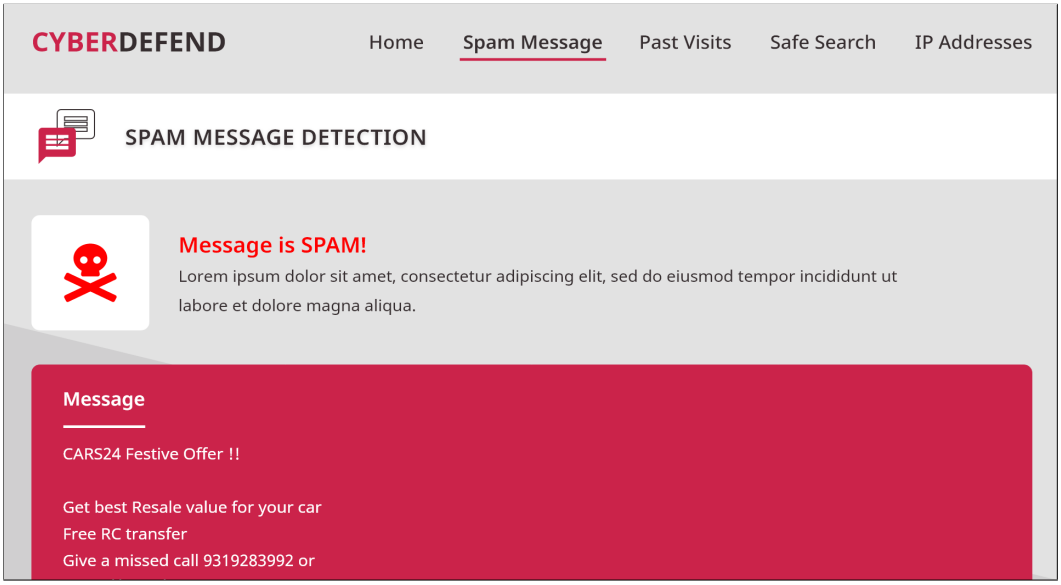


Figure 5.8: Spam Messages: Spam

Screen Description: fig 5.8 shows if the message is spam, a message informing the same user is displayed.

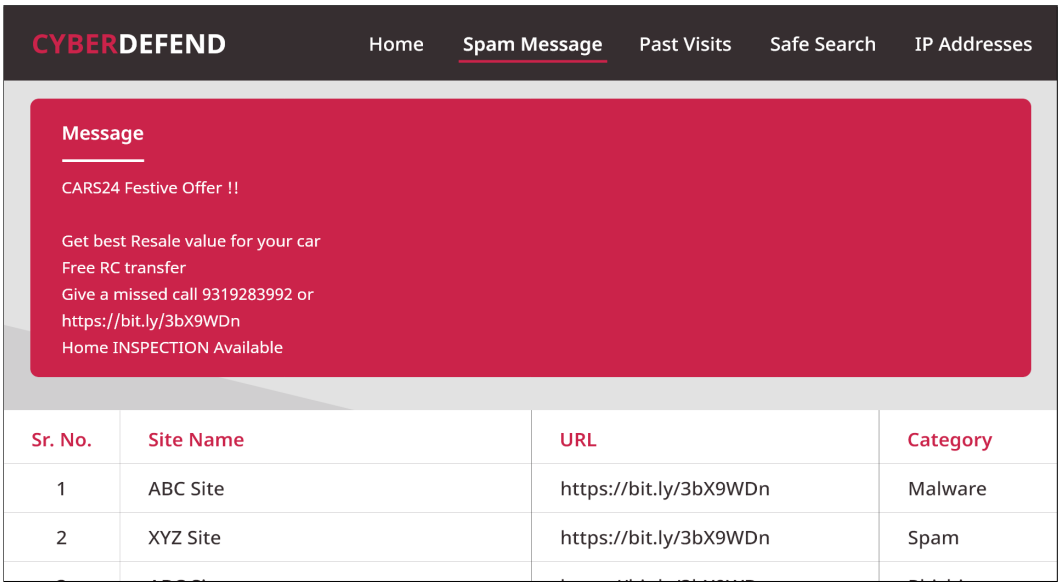


Figure 5.9: Spam Messages: URL List

Screen Description: fig 5.9 shows the URLs which have been a part of the message are also analyzed. Results are displayed in a tabular format as shown.

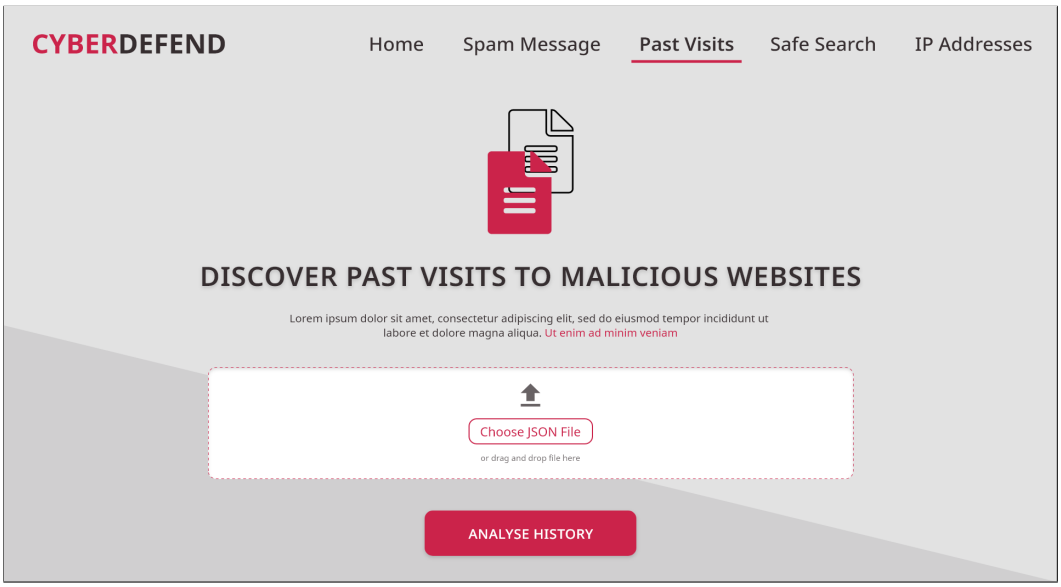


Figure 5.10: Past Visit: Initial

Screen Description: fig 5.10 shows the user is provided a button to choose a JSON file from his device. Alternatively, the user can drag and drop the file. When the "Analyse history" button is clicked, the file is sent to the server-side for extracting the URLs.

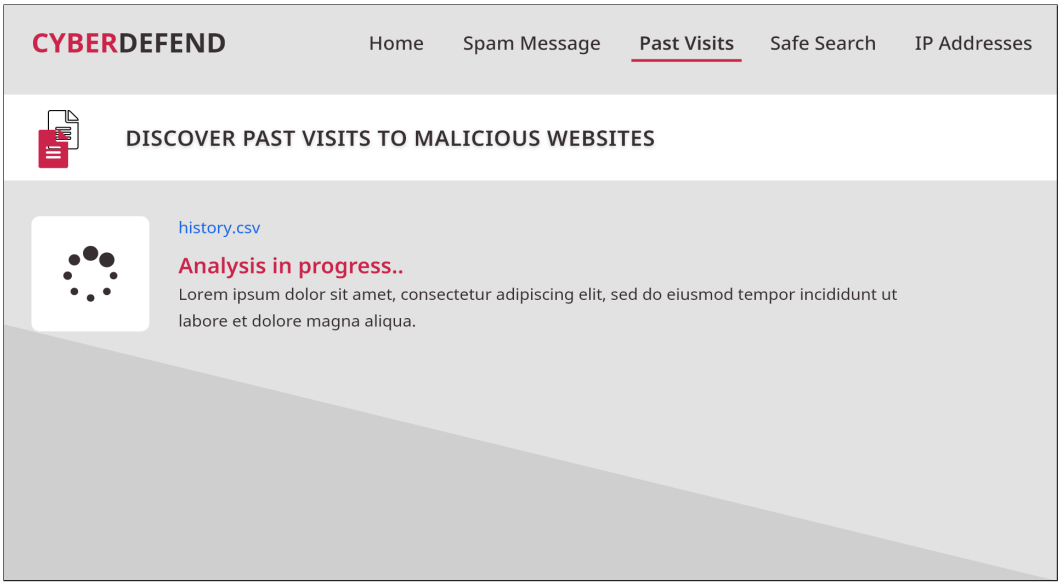


Figure 5.11: Past Visit: Analysis

Screen Description: fig 5.11 shows while the URLs are being analyzed, a message saying that the analysis is in progress is displayed.

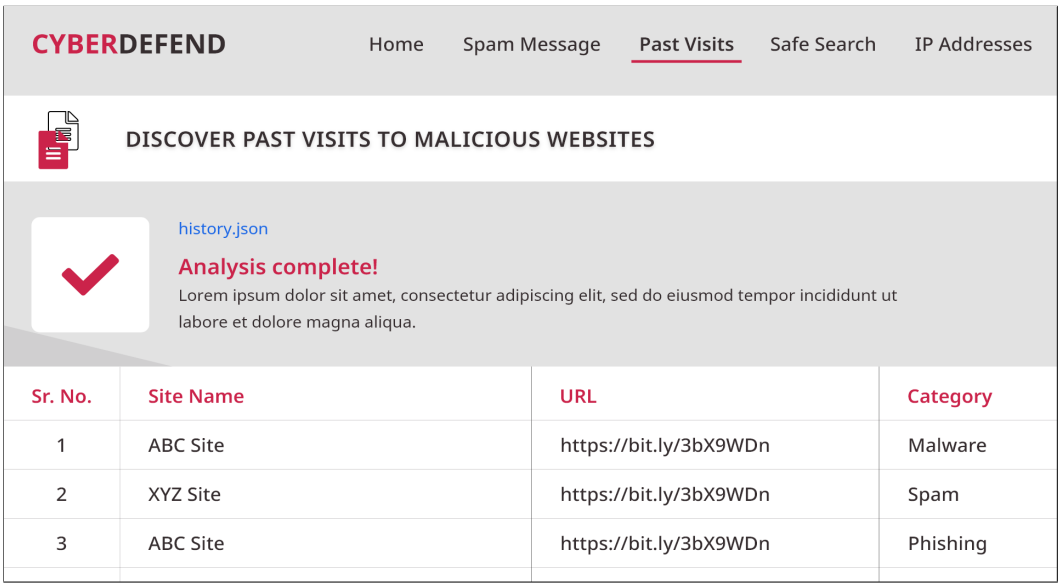


Figure 5.12: Past Visit: Analysis Complete

Screen Description: fig 5.12 shows When the analysis is complete, the results are displayed in a tabular format as shown.

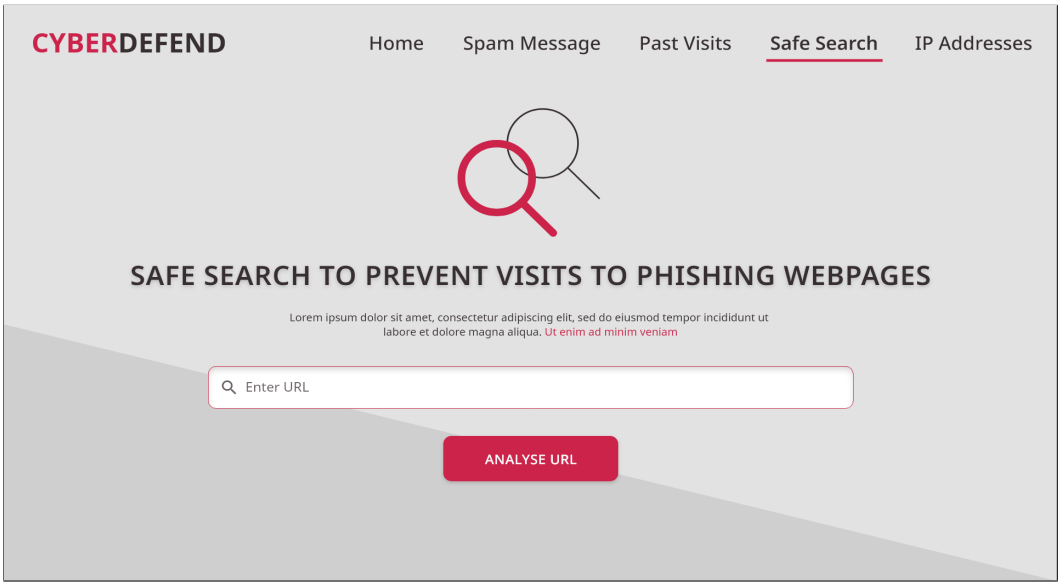


Figure 5.13: Safe Search: Initial

Screen Description: fig 5.13 shows a textbox is provided to the user for entering the URL which needs to be analyzed. Then the user clicks on the "Analyse URL" button, the URL is sent to the server-side for analysis.

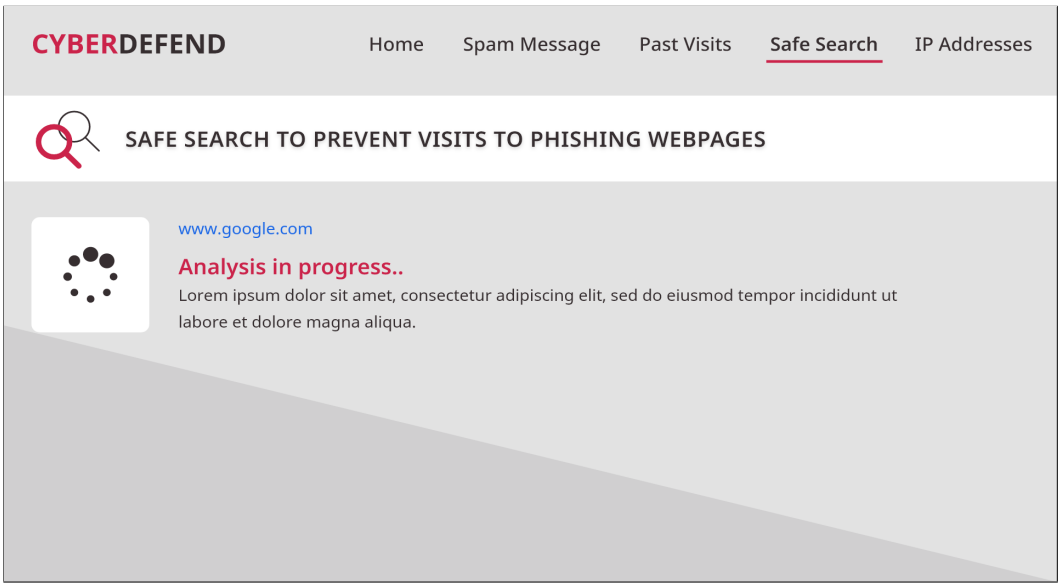


Figure 5.14: Safe Search: Analysis

Screen Description: fig 5.14 shows while the URL is being analyzed, a message saying that the analysis is in progress is displayed.

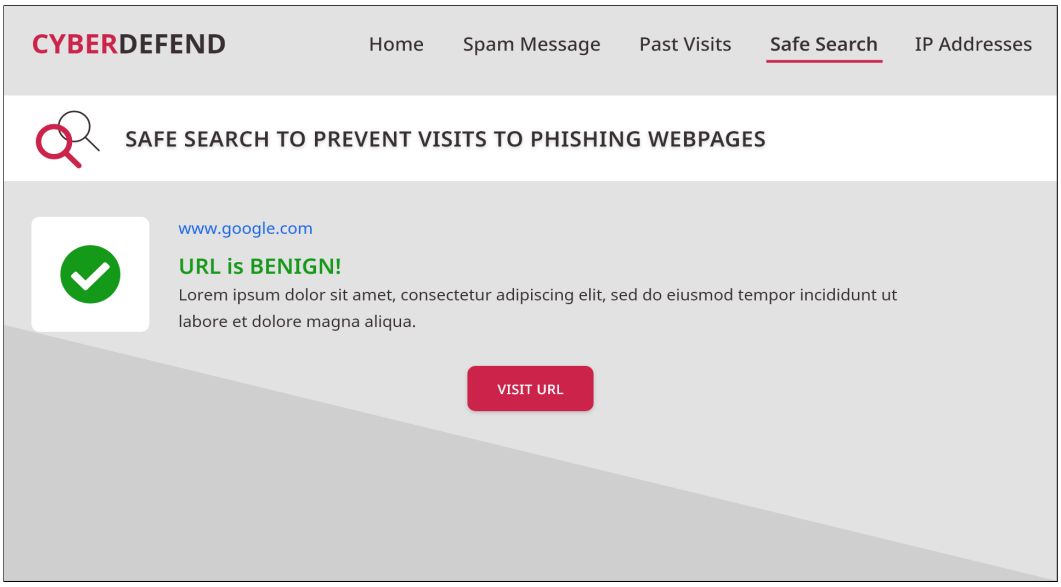


Figure 5.15: Safe Search: Safe URL

Screen Description: fig 5.15 shows If the URL is benign, a "Visit URL" button is displayed. If the user clicks on the same, the user is directed to the particular web page.

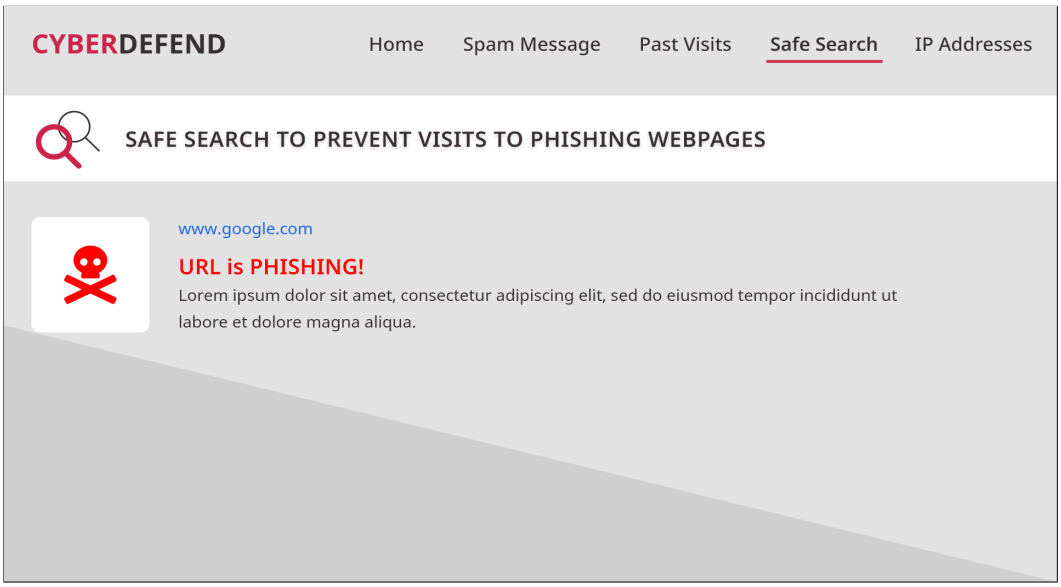


Figure 5.16: Safe Search: Phishing URL

Screen Description: fig 5.16 shows If the URL is phishing/spam/-malicious, a message providing the category of the URL is displayed.

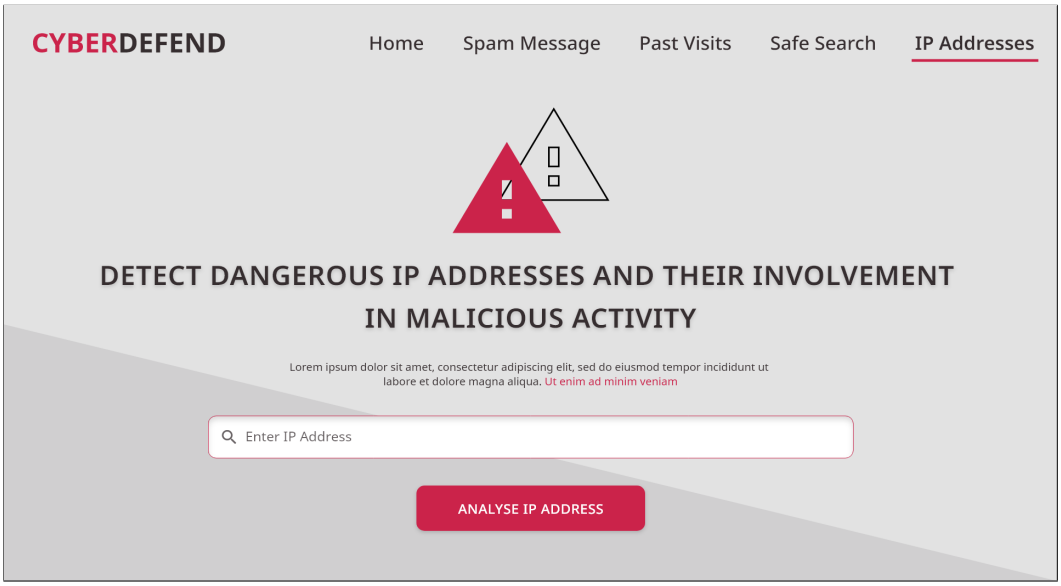


Figure 5.17: IP address: Initial

Screen Description: fig 5.17 shows a textbox is provided to the user for entering the IP address which needs to be analyzed. When the user clicks on the "Analyse IP address" button, the IP address is sent to the server-side where API calls are made.

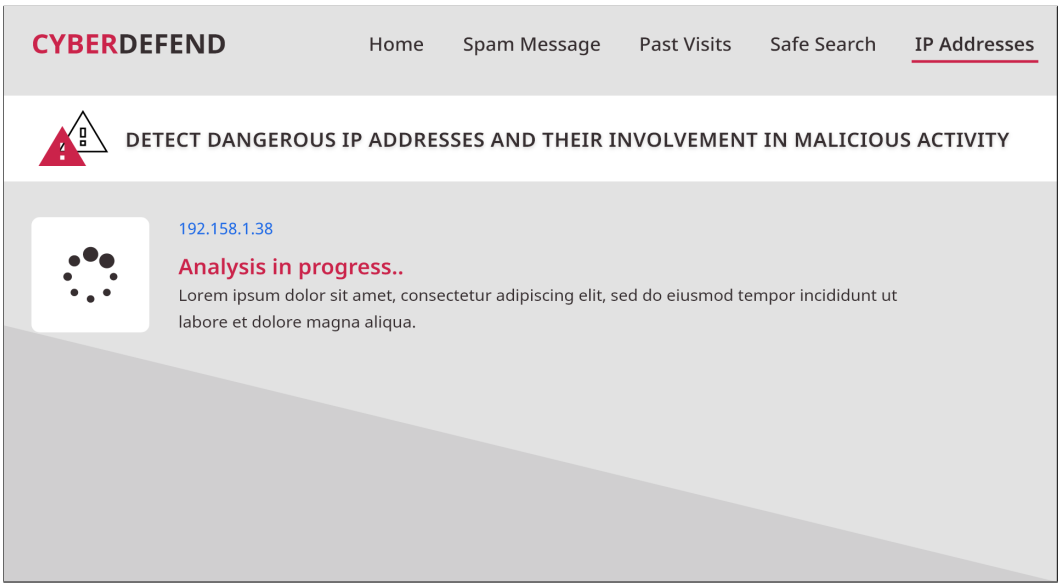


Figure 5.18: IP address: Analysis

Screen Description: fig 5.18 shows while the IP address is being checked against a blacklist, a message saying that the analysis is in progress is displayed.

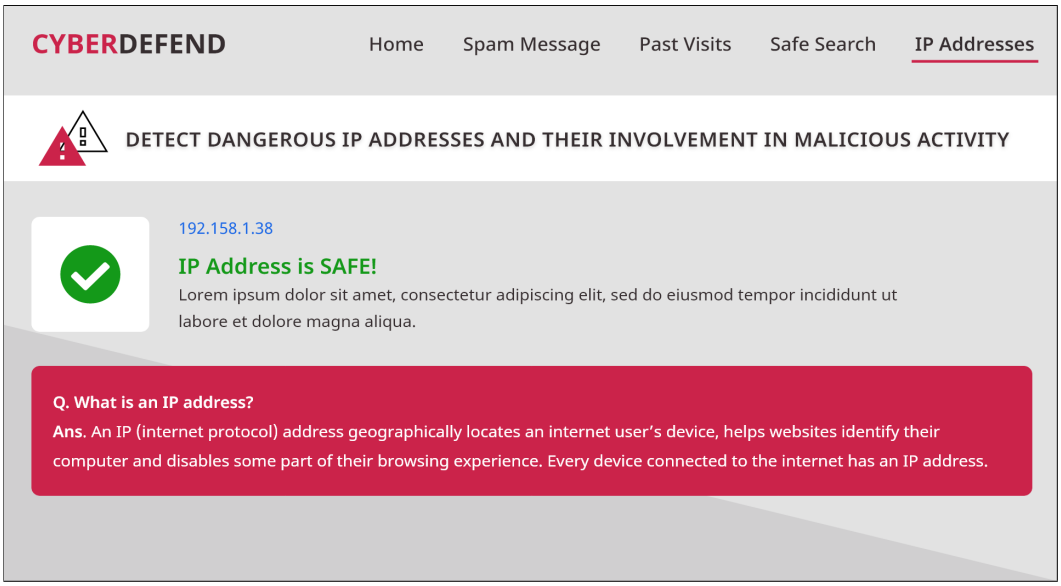


Figure 5.19: IP address: Safe

Screen Description: fig 5.19 shows if the IP address is not blacklisted, the message informing the same is displayed.

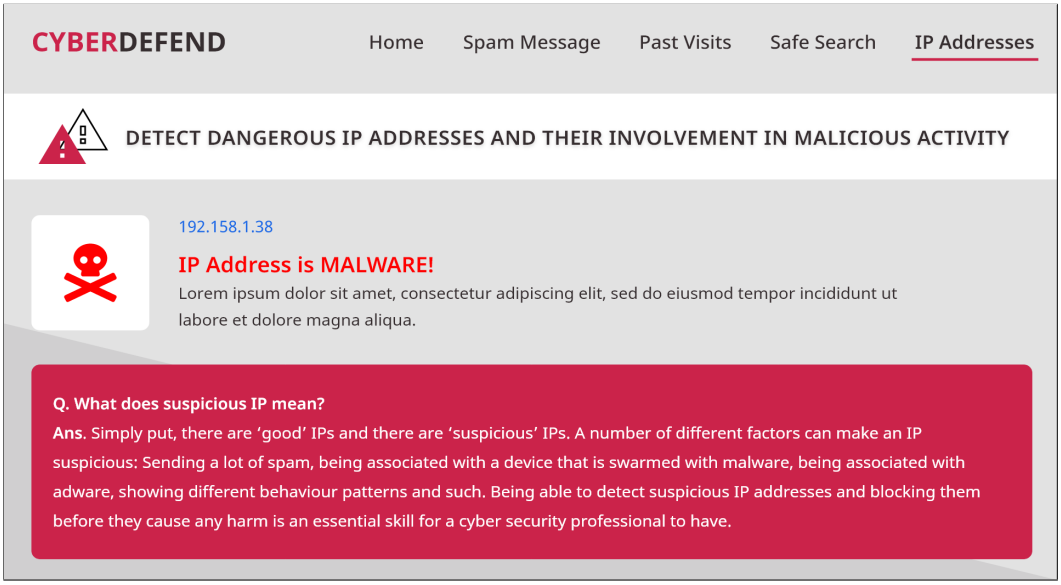


Figure 5.20: IP address: Dangerous

Screen Description: fig 5.20 shows if the IP address is blacklisted, a message informing the same, along with the type of malicious activity the IP address is involved in, is displayed.

5.5 System Architecture

5.5.1 Use Case 1

Use Case 1 is shown in fig 5.21.

Use Case ID	1		
Use Case Name	Spam Classification		
Created by	Gopalkrishna Waja	Last Updated By	-
Date Created	14/10/2021	Date Last Updated:	-

Primary Actor	User
Secondary Actor	WHOIS, Google SafeBrowsing
Description	The user can use the system to identity spam messages with or without URLs by pasting the message in the text box provided on the webpage and allowing the system to classify the message as spam/non-spam and the URL as Safe/Phishing/Malware/Spam
Triggers	Click analyse button
Preconditions	Valid message pasted in the text box
Postcondition	-
Normal Flow	1. Land on "Spam Message" page. 2. Enter/Paste message with or without URL 3. Click Analyse button. 4. Wait for the system to analyse the message and URL(s). 5. Receive message class and display URL class (if any) in tabular format.
Alternative Flow	-
Exception	1. JavaScript code entered instead of text message. 2. Blank input. 3. API not responding (Request Limit reached). 4. Server down.
Includes	-
Optional/Extends	Extract URL features, Classify URLs
Priority	High
Frequency of use	High

Figure 5.21: Use Case 1

5.5.2 Use Case 2

Use Case 2 is shown in fig 5.22.

Use Case ID	2		
Use Case Name	Discover Past visits to malicious websites		
Created by	Charmee Mehta	Last Updated By	-
Date Created	14/10/2021	Date Last Updated:	-

Primary Actor	User
Secondary Actor	WHOIS, Google SafeBrowsing
Description	The user can use the system to identity past visits to malicious websites by uploading his web history and then allowing the systems to classify the URLs in the history as Safe/Phishing/Malware/Spam
Triggers	Click analyse button
Preconditions	Browsing history in valid json format uploaded
Postcondition	-
Normal Flow	1. Land on “past visits” page. 2. Uploads browsing history in valid json format. 3. Click Analyse button. 4. Wait for the system to analyse the uploaded history. 5. Receive results in a tabular format.
Alternative Flow	-
Exception	1. History uploaded in invalid format. 2. No file uploaded. 3. API not responding (Request Limit reached). 4. Server down.
Includes	Extract URL features, Classify URL
Optional/Extends	-
Priority	High
Frequency of use	High

Figure 5.22: Use Case 2

5.5.3 Use Case 3

Use Case 3 is shown in fig 5.23.

Use Case ID	3		
Use Case Name	Safe Search		
Created by	Gaurang Patil	Last Updated By	-
Date Created	14/10/2021	Date Last Updated:	-

Primary Actor	User
Secondary Actor	WHOIS, Google SafeBrowsing
Description	The user can use the safe search feature to verify the safety of an individual URL by allowing the system to classifying the URL as Safe/Phishing/Malware/Spam
Triggers	Click analyse button
Preconditions	Valid URL entered in the text box
Postcondition	-
Normal Flow	1. Land on "Safe search" page. 2. Enter Valid URL. 3. Click Analyse button. 4. Wait for the system to analyse the URL. 5. If URL is safe button to visit URL is received else the threat class is displayed
Alternative Flow	-
Exception	1. Invalid URL format. 2. Blank input. 3. API not responding (Request Limit reached). 4. Server down.
Includes	Extract URL features, Classify URL
Optional/Extends	-
Priority	High
Frequency of use	High

Figure 5.23: Use Case 3

5.5.4 Use Case 4

Use Case 4 is shown in fig 5.24.

Use Case ID	4		
Use Case Name	Detecting dangerous IP addresses		
Created by	Esha Vats	Last Updated By	-
Date Created	14/10/21	Date Last Updated:	-

Primary Actor	User
Secondary Actor	Neutrino
Description	User enters an IP address to check if it is blacklisted and to understand the kind of malicious activity it is involved in, if any.
Triggers	The user lands on the "Dangerous IP address detection" page and submits an IP address in the textbox.
Preconditions	IP address is submitted in the textbox.
Postcondition	-
Normal Flow	1. User lands on the "Dangerous IP address detection" page 2. User enters valid IP address 3. IP address is checked against a blacklist using Neutrino API at the backend. 4. If the IP address is blacklisted, the same, along with the type of malicious activity associated with the address (if available) is displayed to the user. If the IP address is not blacklisted, then a message informing the same is displayed.
Alternative Flow	
Exception	1. Blank submission in the textbox. 2. Invalid IP address submitted. 3. JavaScript code is submitted instead of IP address. 4. Per day API usage limit is exhausted. 5. API not responding due to some technical issue.
Includes	Checks against a blacklist
Optional/Extends	-
Priority	Medium
Frequency of use	Low

Figure 5.24: Use Case 4

5.6 Structural Diagrams

5.6.1 Component Diagram is shown in fig 5.25

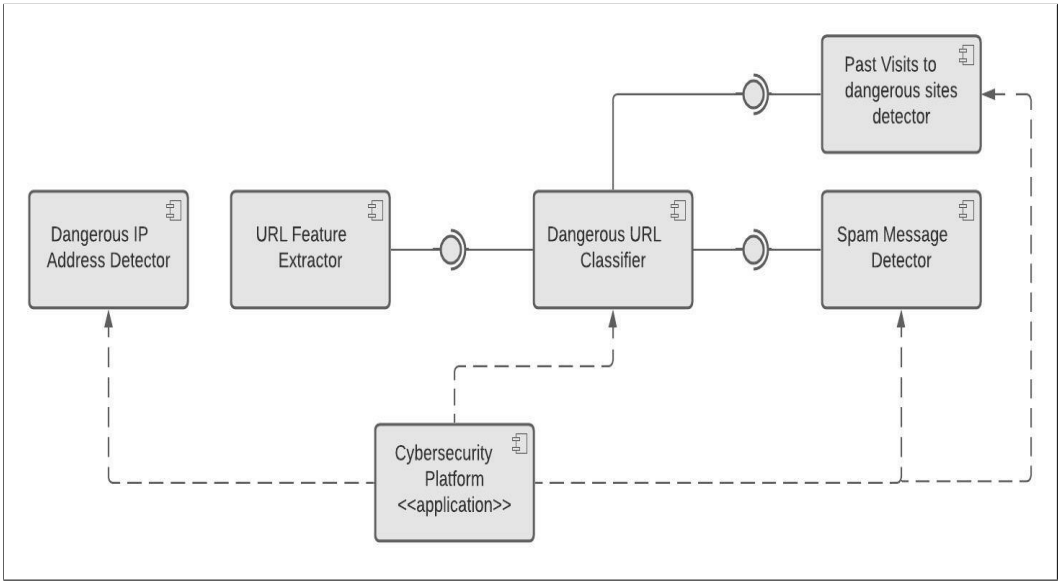


Figure 5.25: Component Diagram for CyberDefend

5.6.2 Deployment Diagram is shown in fig 5.26

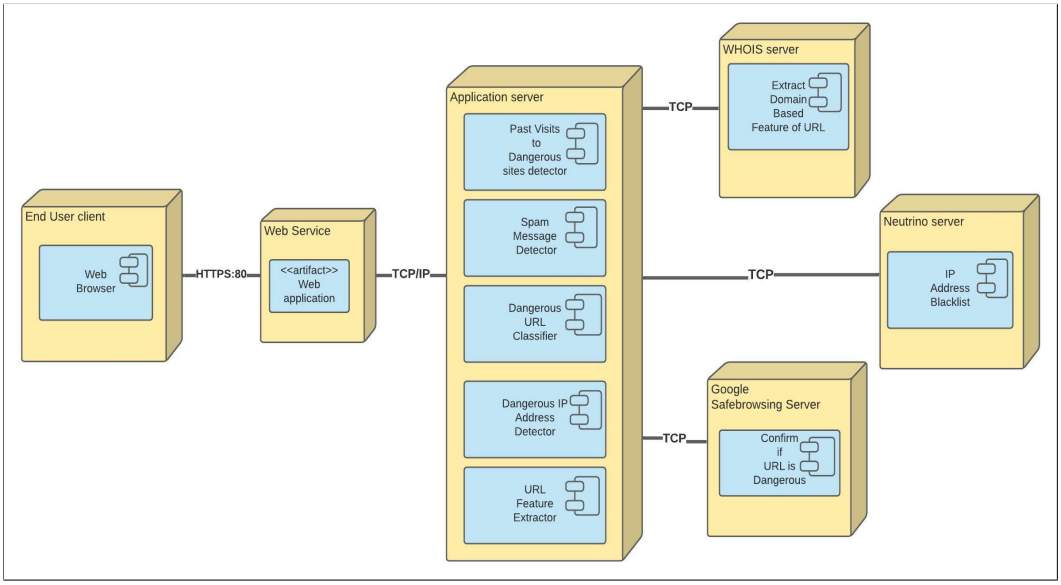


Figure 5.26: Deployment Diagram for CyberDefend

5.7 Behavioral Diagrams

5.7.1 Activity Diagram is shown in fig 5.27

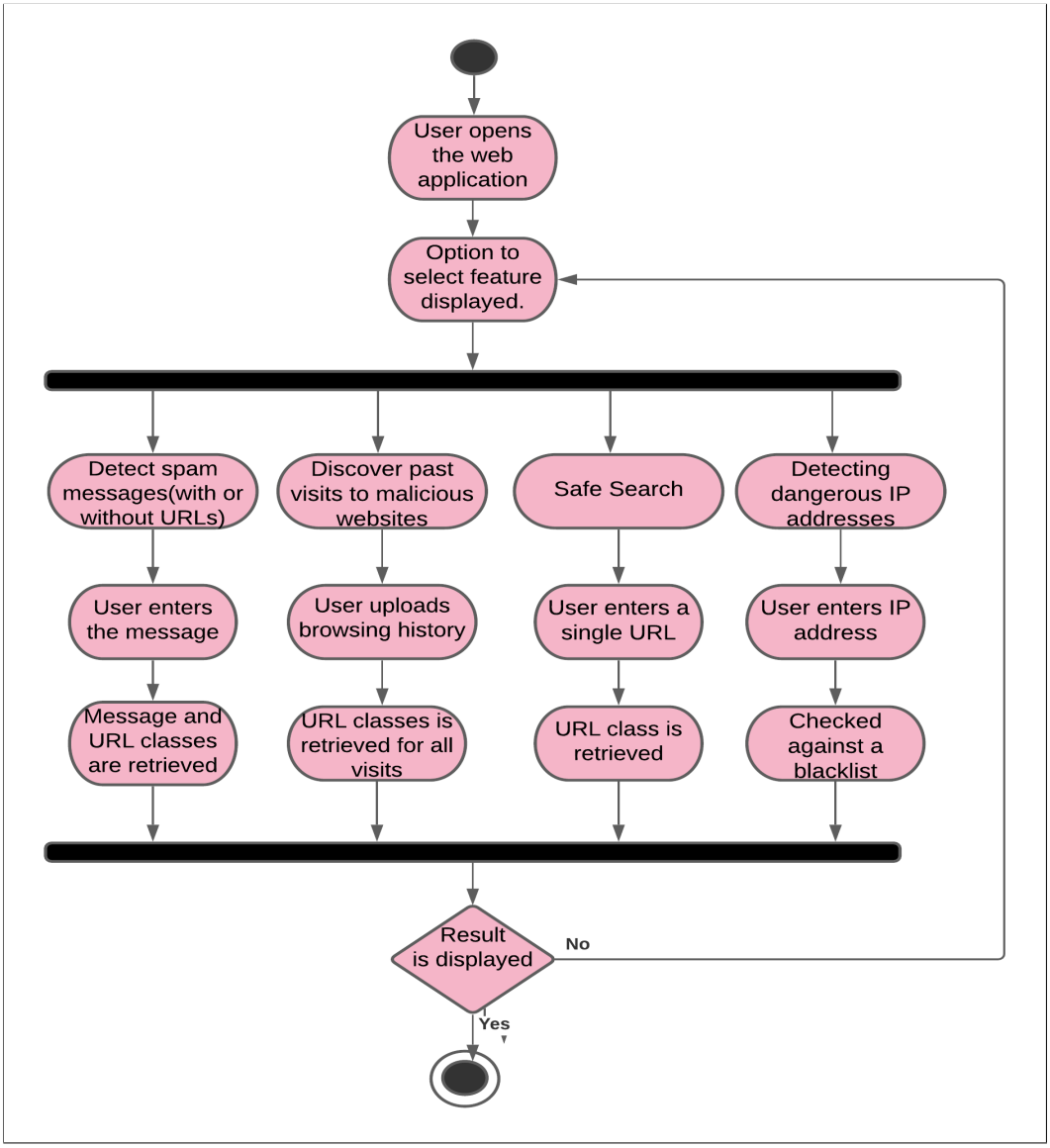


Figure 5.27: Activity Diagram for CyberDefend

5.7.2 Use case diagram is shown in fig 5.28

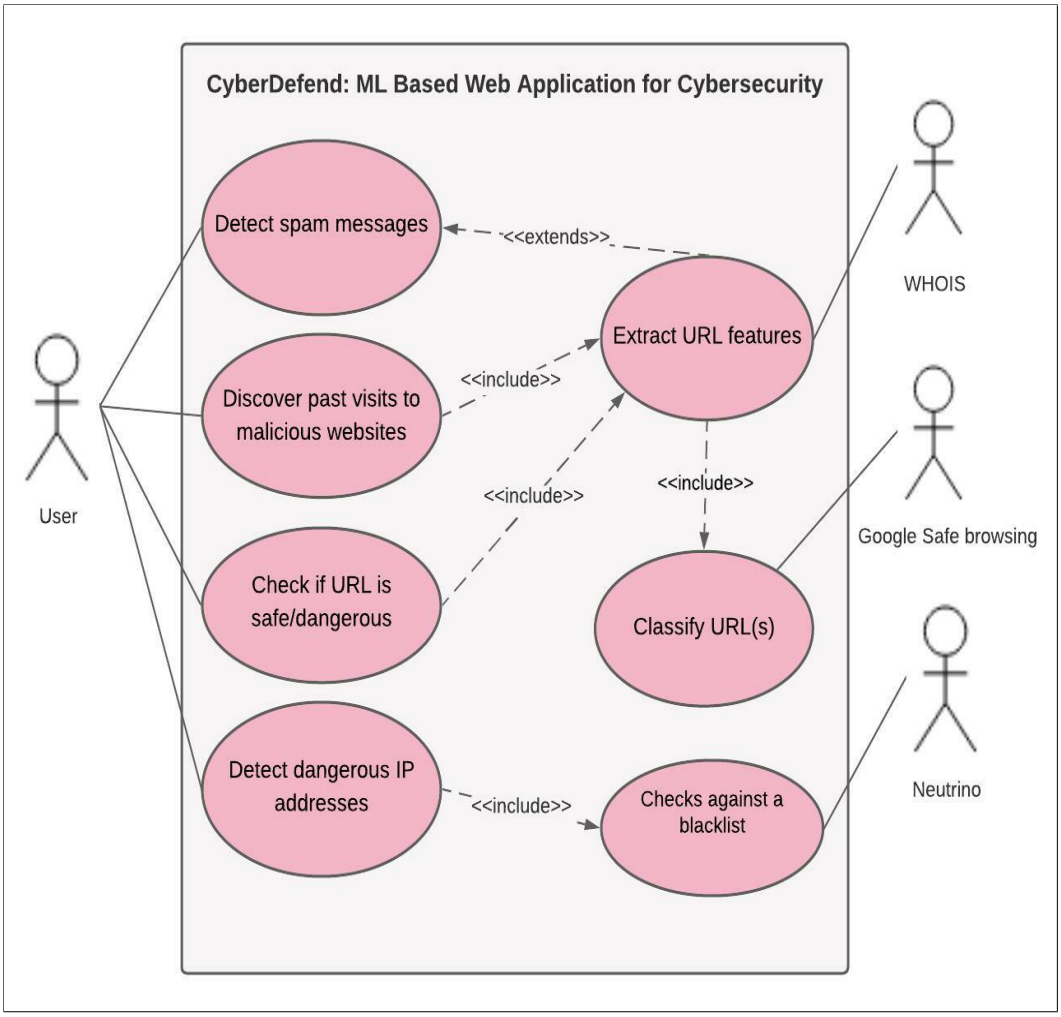


Figure 5.28: Use Case Diagram for CyberDefend

5.8 Interaction Diagrams

5.8.1 Sequence Diagram shown in fig 5.29 to 5.32

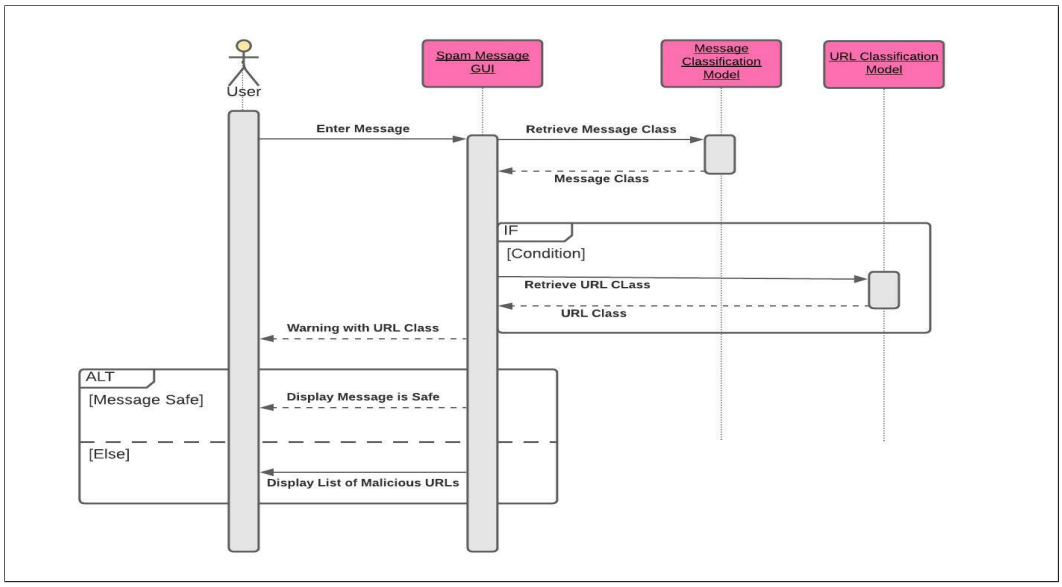


Figure 5.29: Sequence Diagram for Spam Feature

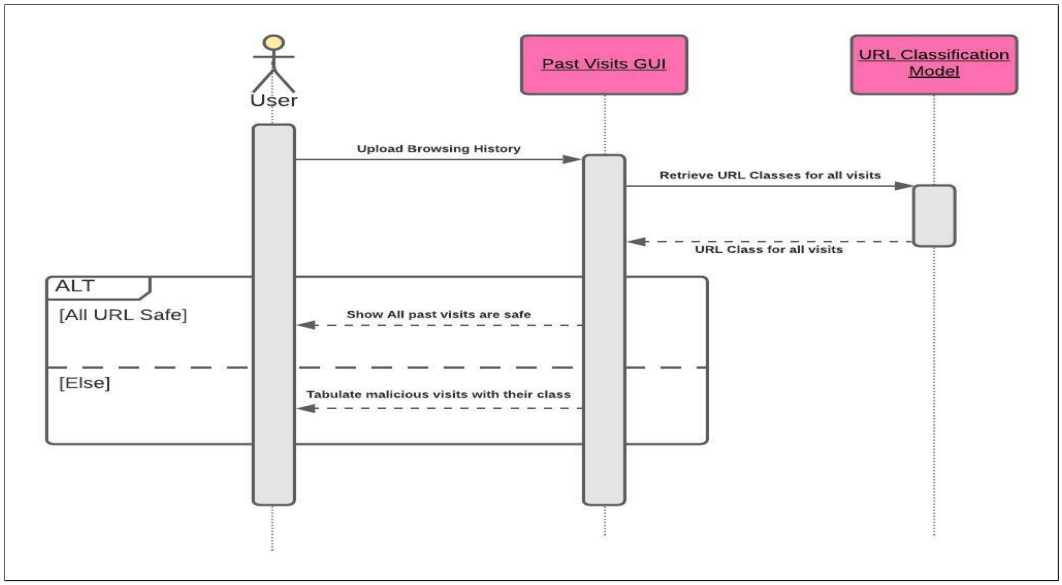


Figure 5.30: Sequence Diagram for Past Visits Feature

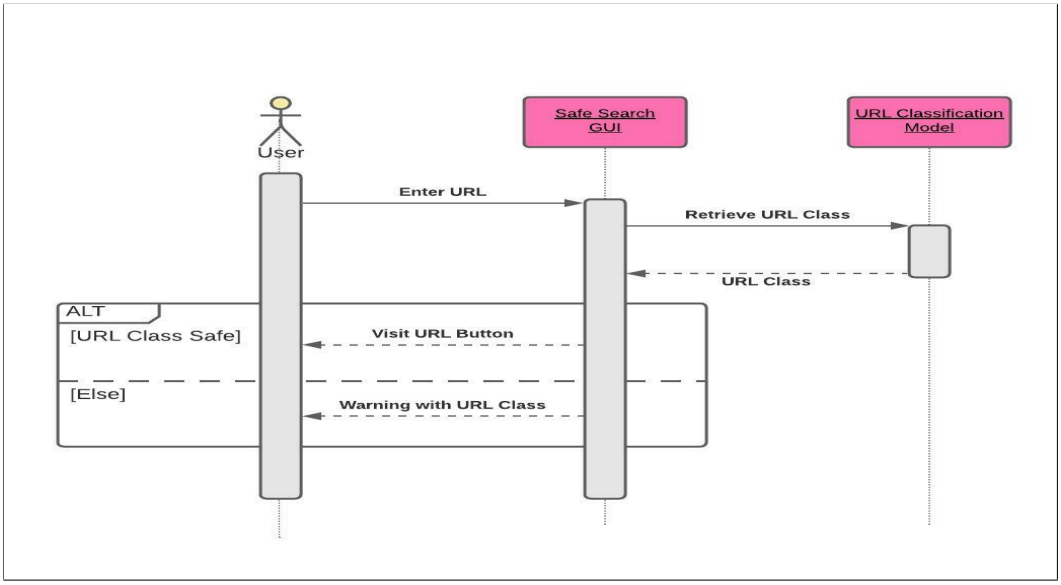


Figure 5.31: Sequence Diagram for Safe Search Feature

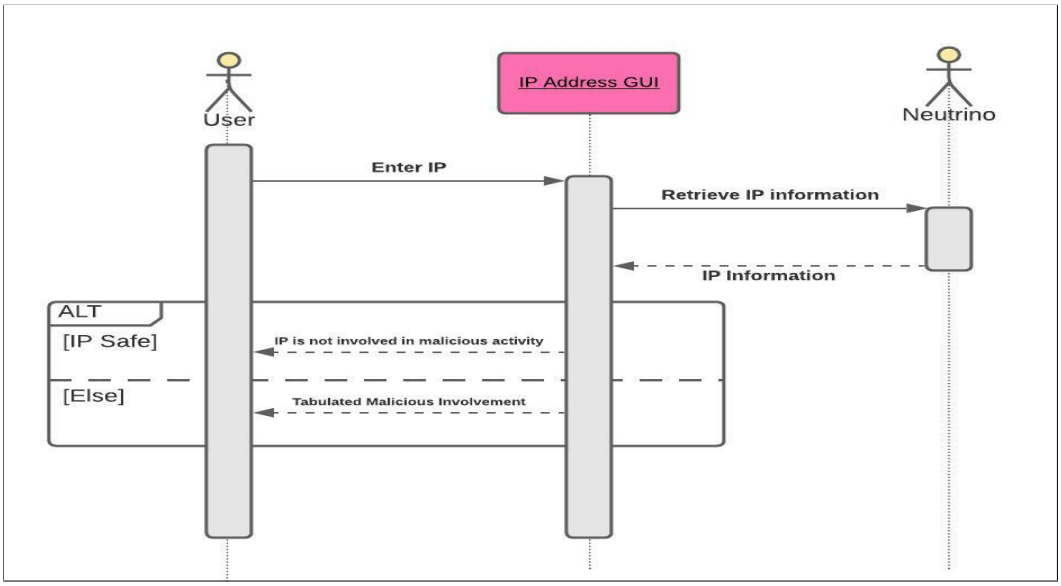


Figure 5.32: Sequence Diagram for IP Address Feature

Summary: The web application uses a client-server architecture. The necessary UML diagrams for designing the software have been looked at in detail. The next chapter deals with the implementation details.

CHAPTER 6

Implementation

This chapter discusses the implementation of the project in detail. It lists down the technology stack used with an algorithmic implementation. A detailed implementation procedure for each of the modules is presented in this chapter.

6.1 Introduction

In this chapter, we have included the implementation details of the CyberDefend Application. We have included details like technologies and algorithms used, project structure, and images of different pages of the website. It must be noted that although in the previous chapters i.e. SPMP, SRS, and SDD we have provided the details of the Dangerous IP address module, upon discussion with our guide and mentor, we have decided not to implement it due to the unexpected withdrawal of a team member (Esha Vats) from the project.

6.2 Technologies Used

The Technology Stack used for the implementation of CyberDefend is shown in table 6.1 as follows:

Front End	Back End	ML Models
HTML	Streamlit	Python
CSS	GoogleSafe Browsing API	Google Colaboratory
JavaScript	-	-
Bootstrap	-	-

Table 6.1: Technology Stack

6.3 Dataset Description

For Spam message detection pre-existing open-source datasets often have a limited variety of text messages and hence the dataset which we have used is a combination of a pre-existing dataset and a self-created dataset formed by labeling messages collected from our mobile phones over 6 months. The self-collected messages include a variety of banking, e-commerce, and other types of messages. The pre-existing dataset is UCI's public dataset containing 5169 unique labeled SMSs out of which, as visible in fig.1, 653 are spam and 4516 are non-spam. In addition to this our self-collected dataset contains 3582 unique messages with 2429 spam and 1153 non-spam messages. So, cumulatively the final dataset contains 8751 messages out of which 3082 are spam and 5669 are non-spam. The dataset summary is available in tables 6.2 and 6.3.

Dataset/Type	Spam	Non-Spam	*Total*
UCI Dataset	653	4516	5169
Self-Collected	2429	1153	3582
Total	3082	5669	8751

Table 6.2: Dataset Description for Spam Classifier

For URL Classifier we have used a dataset of 48502 URLs where the instances for the four classes have been collected from different sources

as mentioned in the table below.

CATEGORY	SOURCE	NUMBER	TOTAL
Safe	DMOZ	15,000	
Spam	UNB	11,999	
Phishing	PhishTank	11,500	
Malware	URLhaus	10,003	48502

Table 6.3: Dataset Description for Spam Classifier

6.4 Algorithm

6.4.1 Spam Message Classifier

For the implementation of the Spam Classifier, We have used the Multi-Channel CNN architecture. As shown in fig 6.1 The first layer of this model is the Embedding layer where we are using the Word2Vec, Random, and GloVe Embeddings for finding the word vector representation of the input SMS message. These three embeddings form the three different channels of the model. For each channel, the word embedding is passed through four convolution layers oriented in parallel to each other with kernel sizes of 2,3,5, and 7 for n-gram level feature extraction. After this, the extracted feature maps are passed through dropout regularization and MaxPooling for dimensionality reduction followed by a flattening operation. The concatenated flattened features are then passed through the feed-forward network for final binary classification.

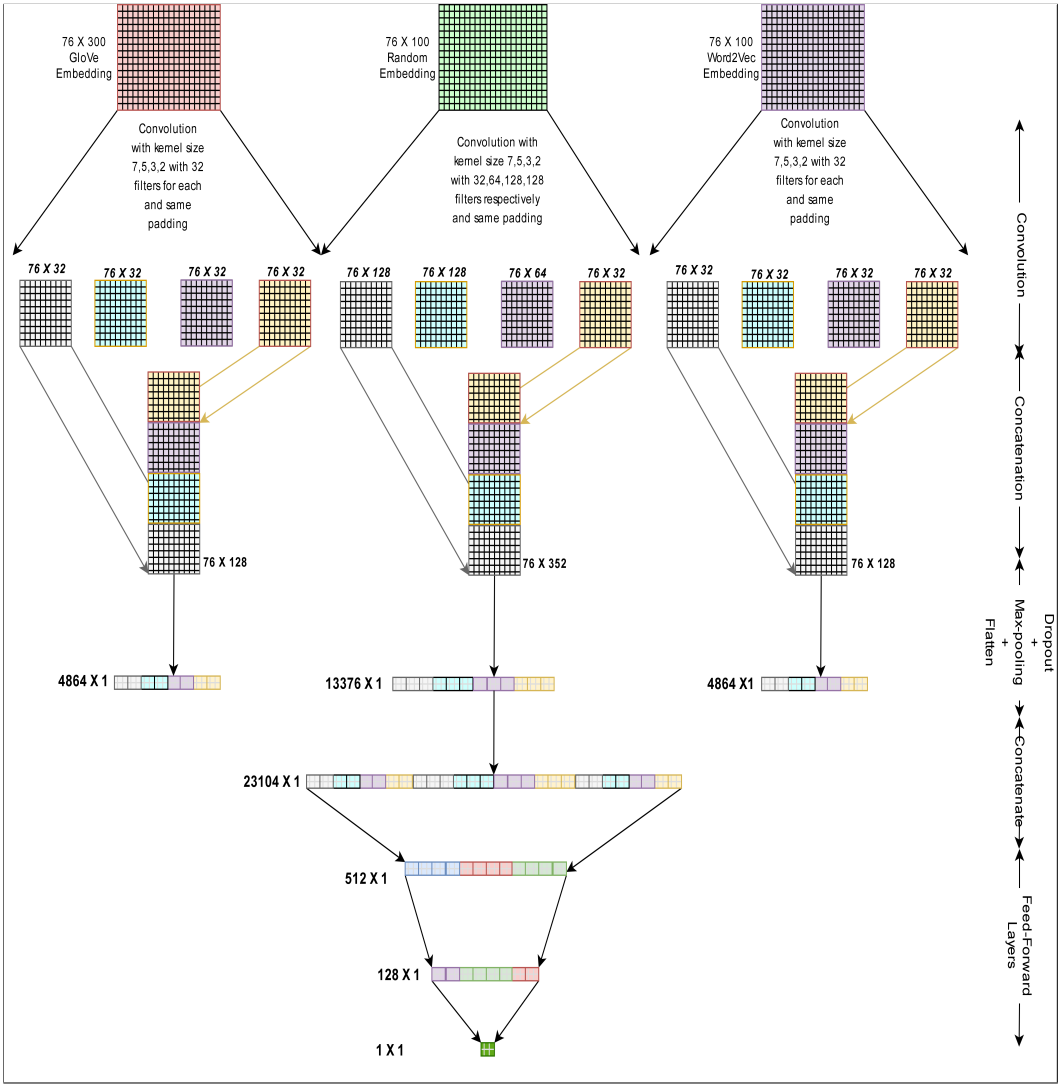


Figure 6.1: Implementation of Spam Classifier

6.4.2 URL Classifier

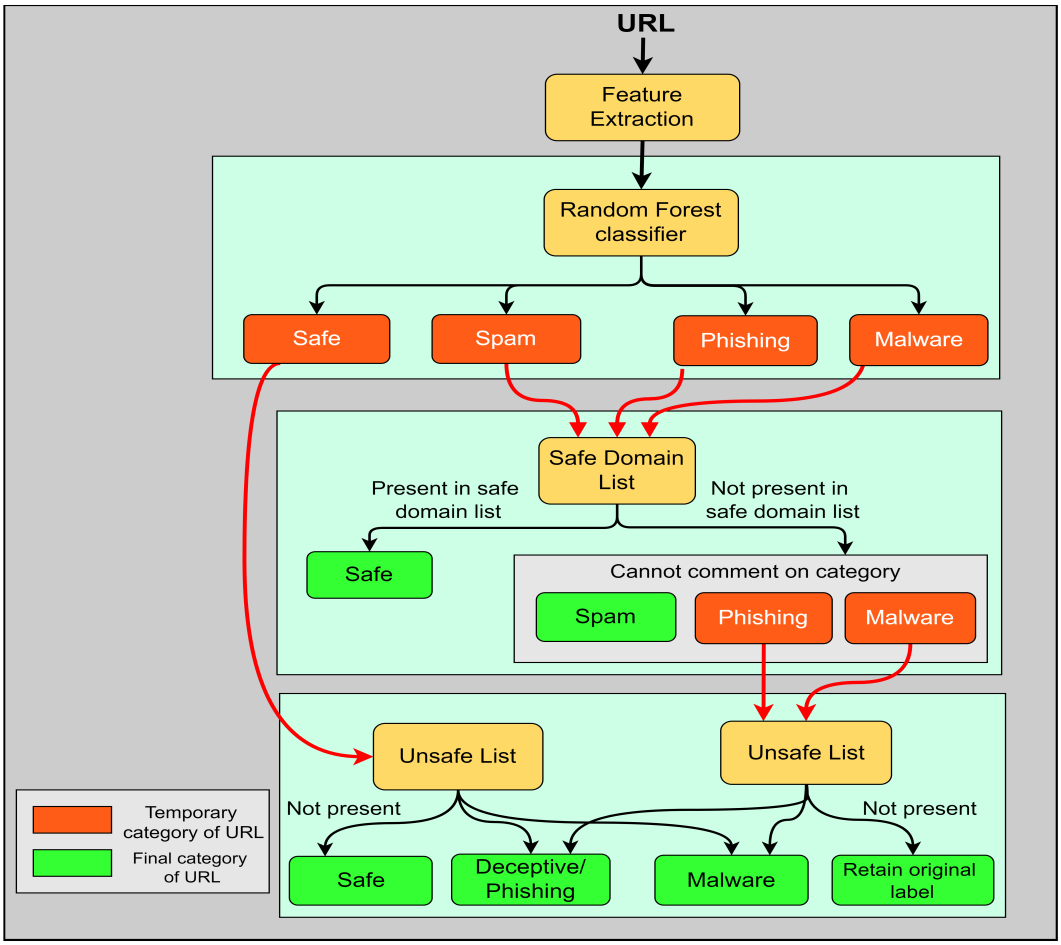


Figure 6.2: Implementation of URL Classifier

As shown in fig 6.2 the URL Classifier takes a user URL as the input and feeds the features of the URL to the Random Forest Classifier which then Classifies the URL into one of the four classes i.e. Safe, Spam, Phishing, Malware. To avoid misclassification we have made use of two lists, the Safe domain list obtained from Cisco's top 1 million URLs and the Unsafe list from the Google SafeBrowsing API. We use these lists to cross-verify the output of the random forest classifier with existing data available about safe and unsafe URLs and over-turn the labels if required.

6.5 Implementation

6.5.1 Project Structure

CyberDefend project makes use of 23 files which are listed below in fig 6.3.

Figure 6.3: Project Structure

6.5.2 Home page

1. Entering the URL: http://cutt.ly/cyberDefend leads us to the following homepage as shown in fig 6.4

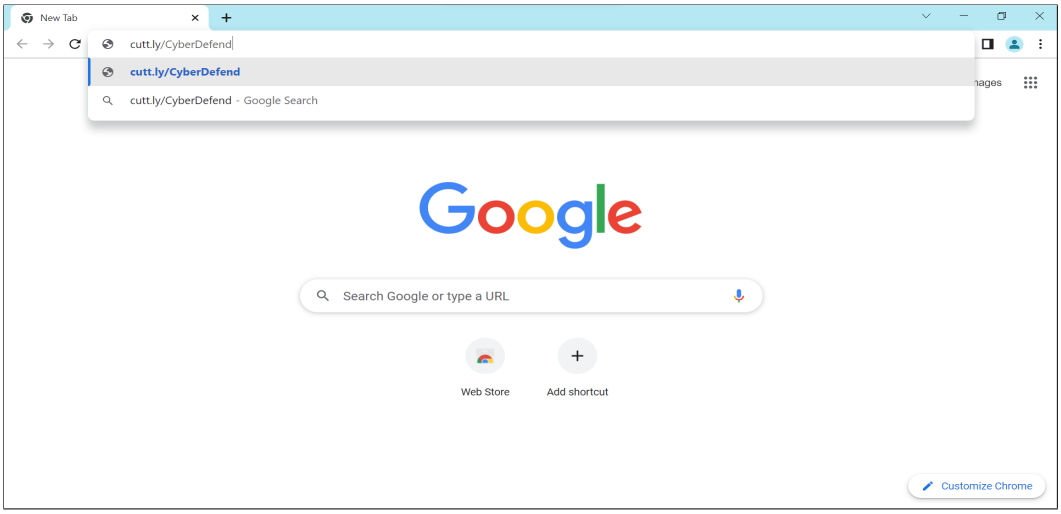


Figure 6.4: Navigate to website

2. Land on the homepage as shown in fig 6.5

Figure 6.5: Homepage

3. List of features on the homepage as shown in fig 6.6

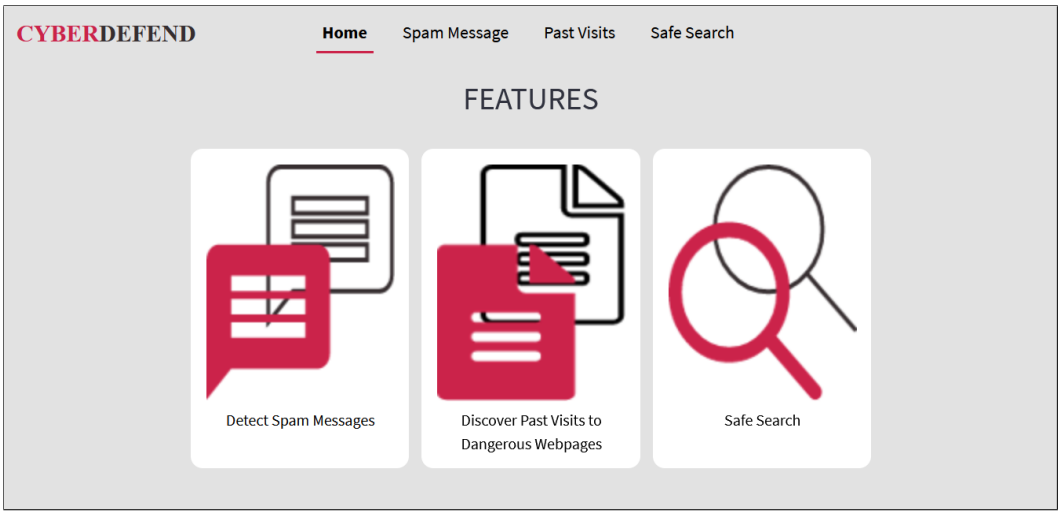


Figure 6.6: Features on homepage

6.5.3 Spam Message Detector

1. Navigate to Spam Message Detector page as shown in fig 6.7

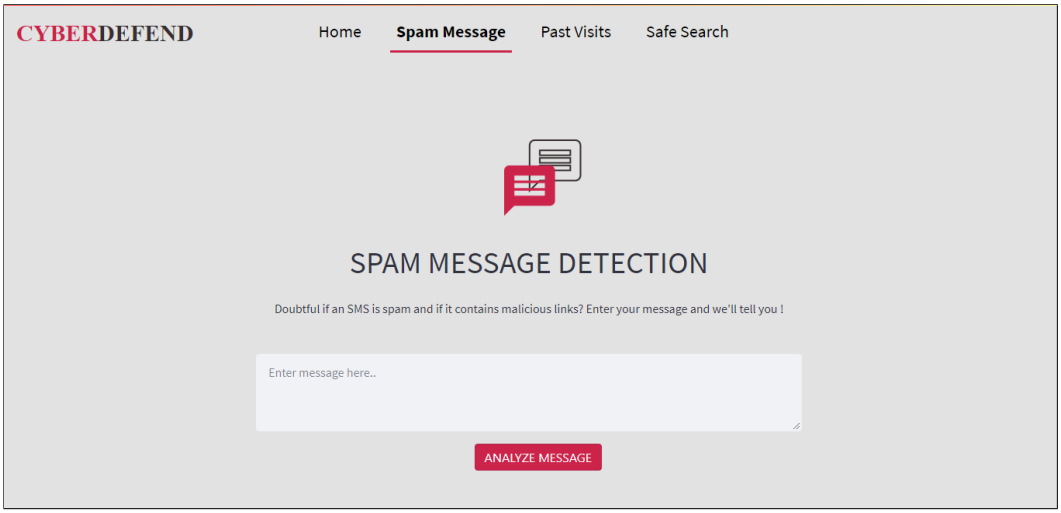


Figure 6.7: Spam message Dectector page

2. Enter a message and click Analyze button as shown in fig 6.8

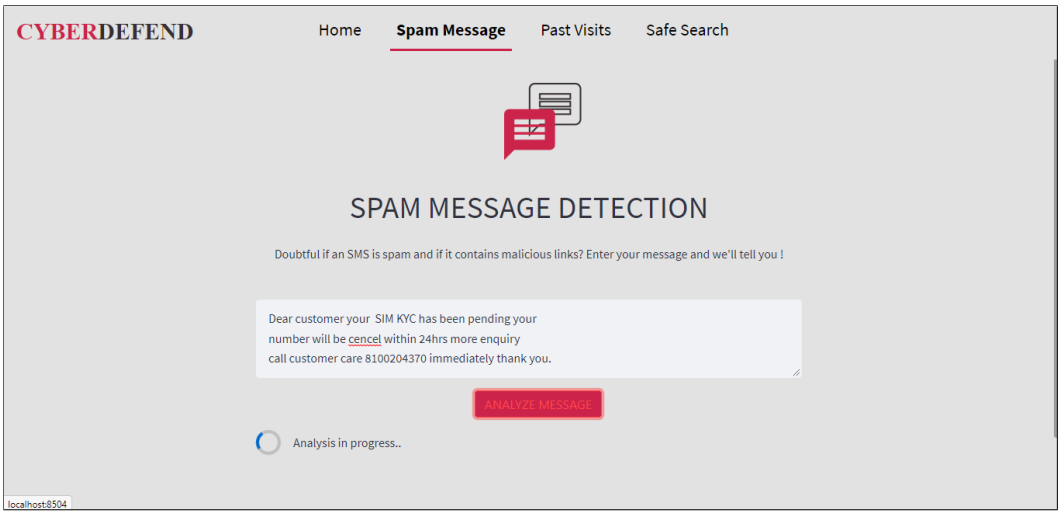


Figure 6.8: Spam message Dectector page

3. Result for Spam message as shown in fig 6.9

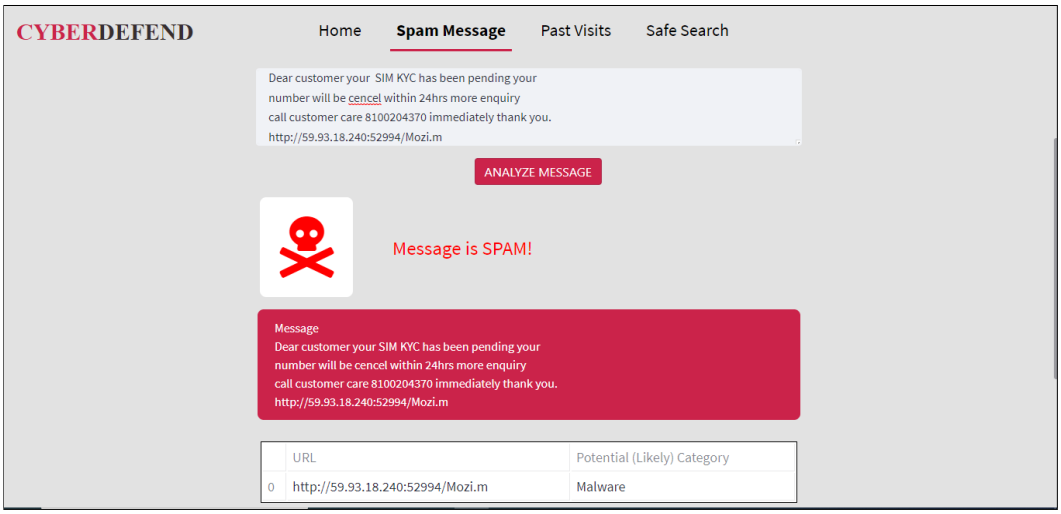


Figure 6.9: Result for Spam Message

4. Result for Safe message as shown in fig 6.10

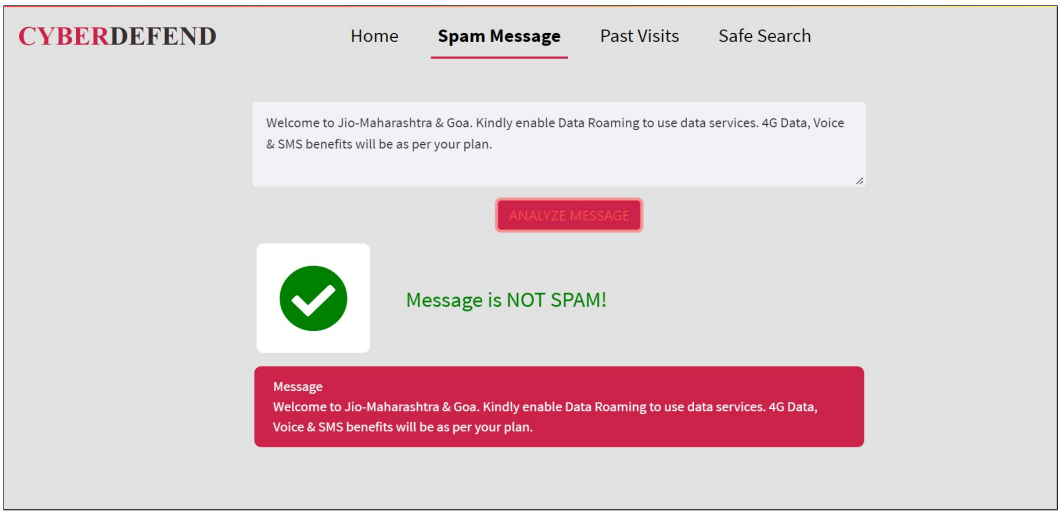


Figure 6.10: Result for Safe Message

6.5.4 Safe Search

1. Navigate to Safe Search page as shown in fig 6.11

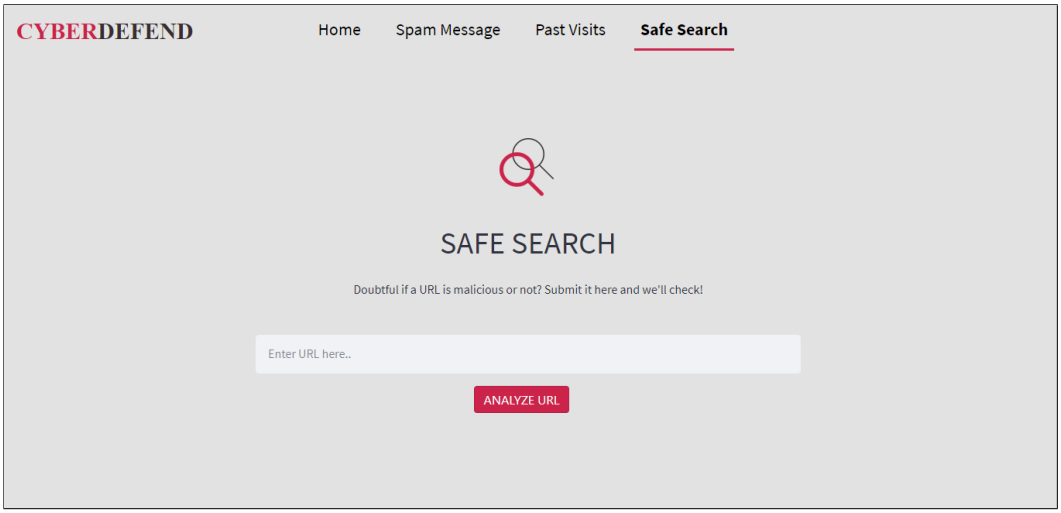


Figure 6.11: Safe Search page

2. Enter Url and Analyze as shown in fig 6.12

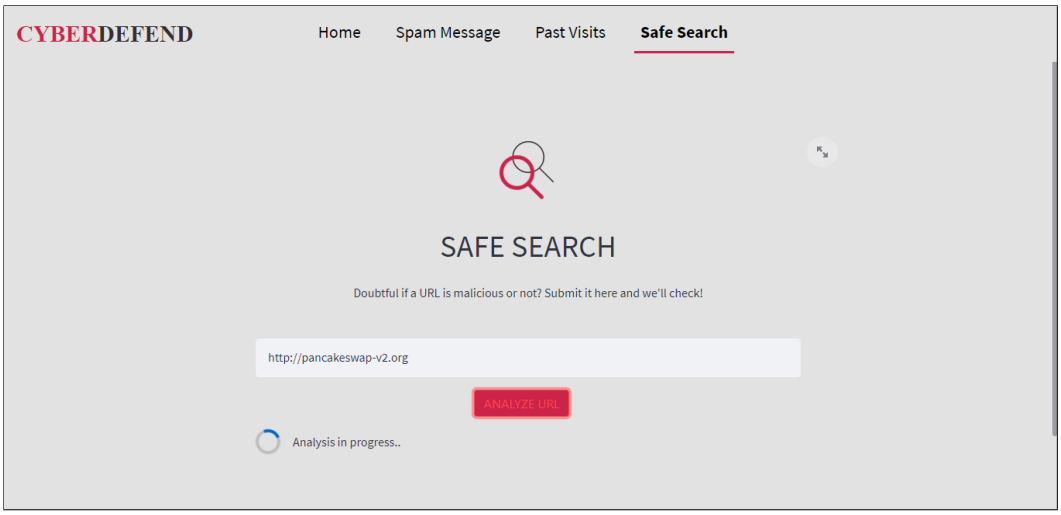


Figure 6.12: Enter URL and Analyze

3. Result for Malicious URL is shown in fig 6.13

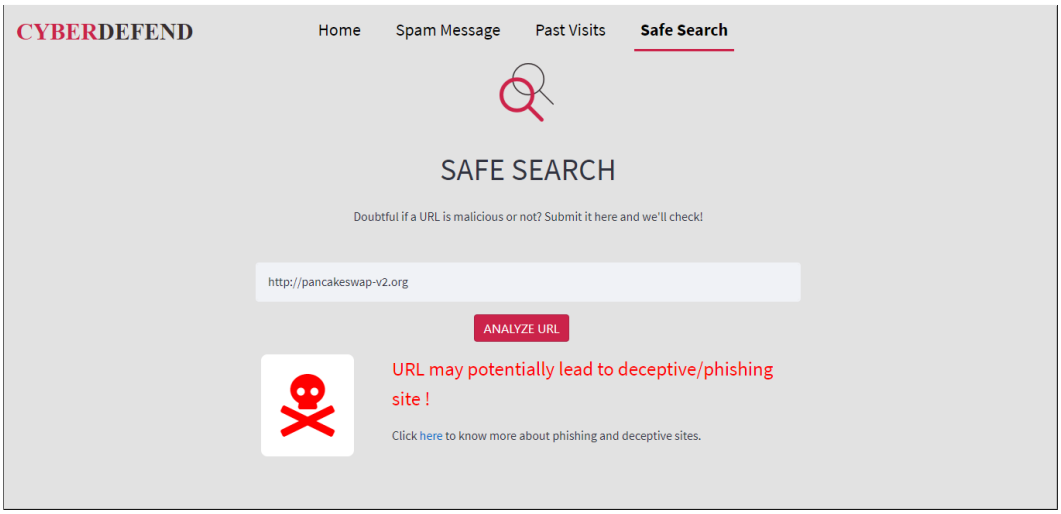


Figure 6.13: Result for Malicious URL

4. Result for Safe URL is shown in fig 6.14

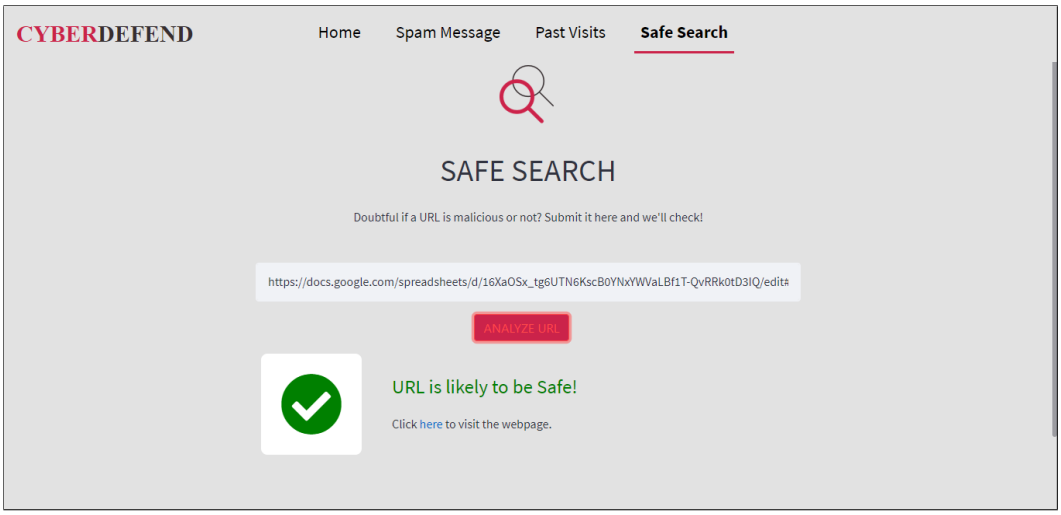


Figure 6.14: Result for Safe URL

6.5.5 Discovery of Past Visits to Malicious Sites

1. Navigate to Past visits page as shown in fig 6.15

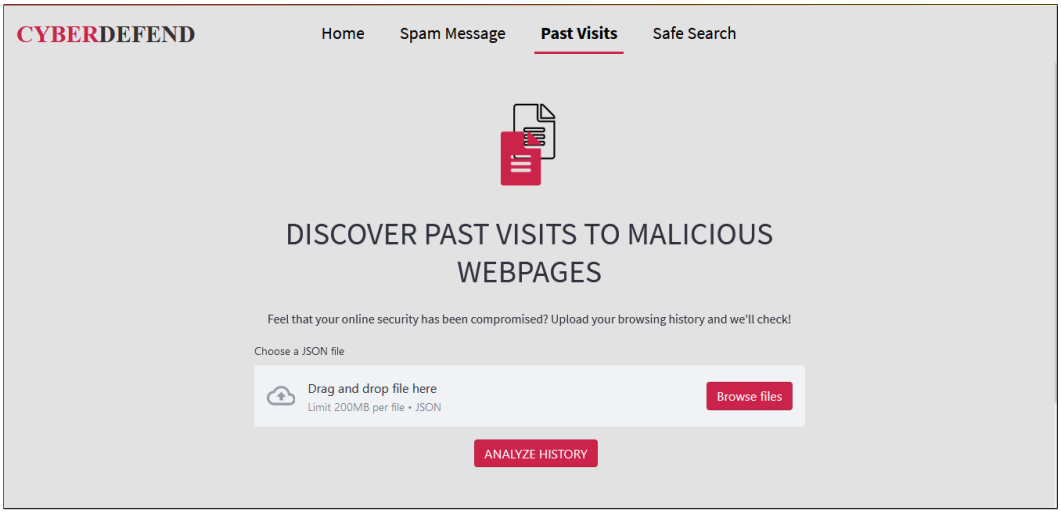


Figure 6.15: Past visits page

2. Upload Browsing History in JSON format is shown in fig 6.16

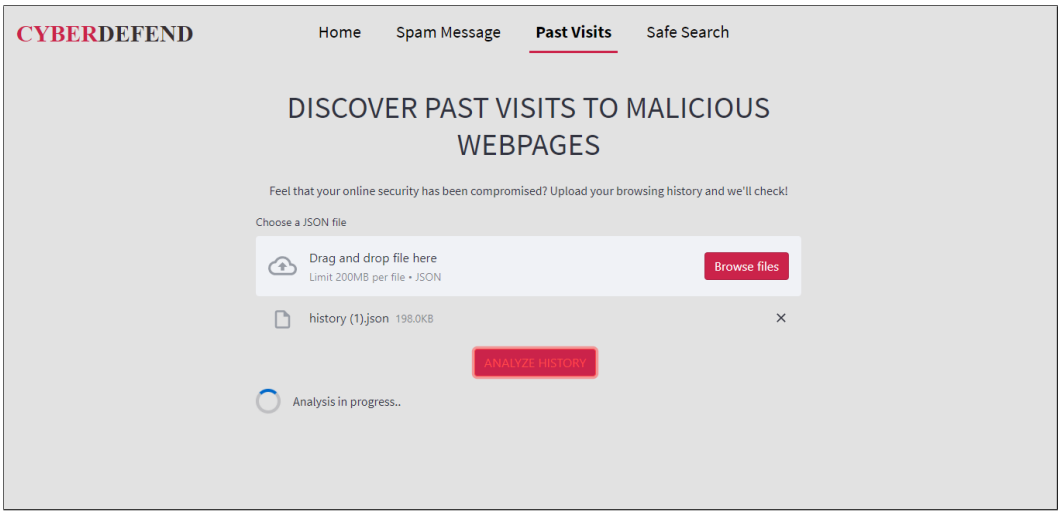


Figure 6.16: Upload History and Analyze

3. Result for the presence of Malicious History is shown in fig 6.17

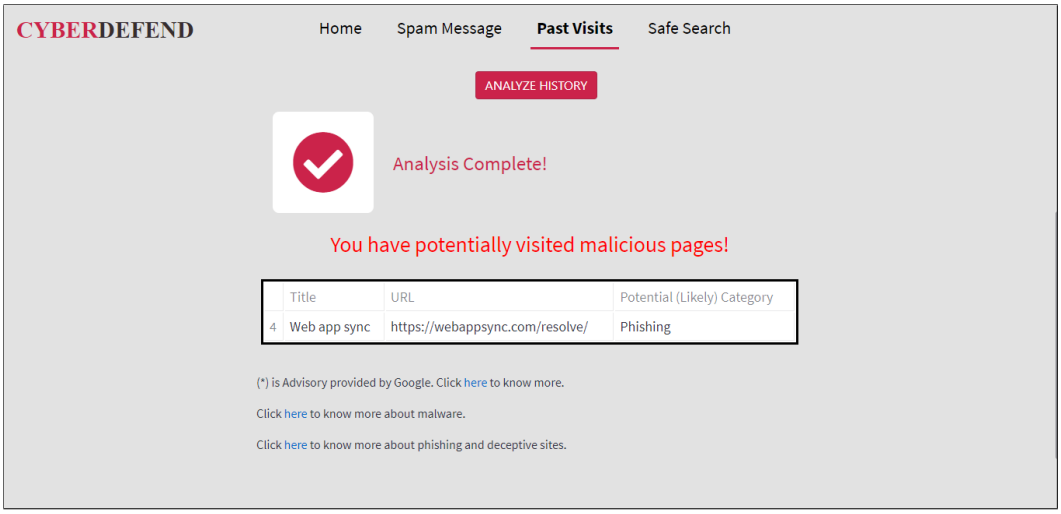


Figure 6.17: Result for malicious history

4. Result for the presence of Safe History is shown in fig 6.18

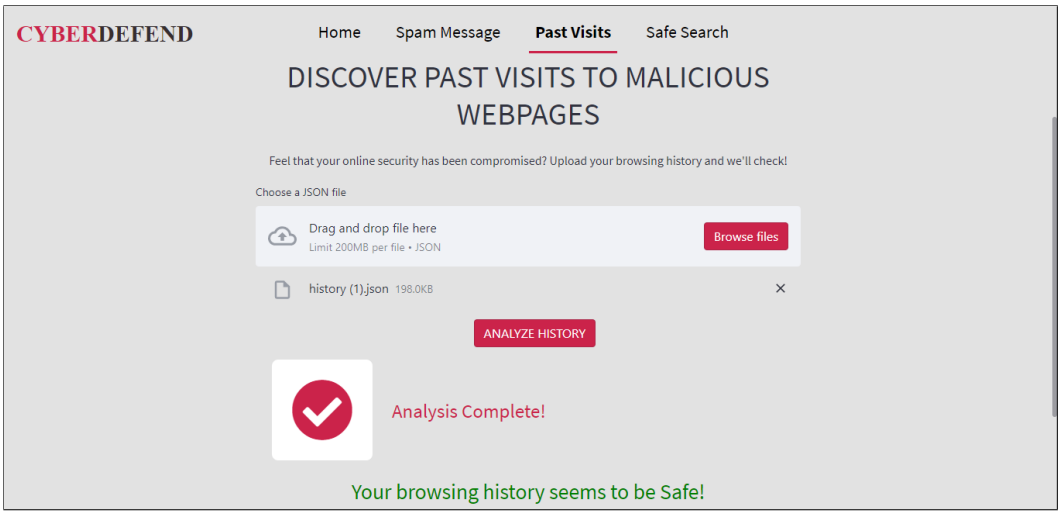


Figure 6.18: Result for Safe history

Summary: This chapter elaborated on the implementation details of the classifiers as well as the web application. User interface implementation details had also been shown. The next chapter elaborates on the testing plan and the test cases required for smooth functioning of the application.

CHAPTER 7

System Test Document

This chapter provides the System Test Document which illustrates the test approach by stating the features to be tested and not to be tested. Testing tools and the environment under which the system will go under test are explained with details about the test cases to be performed.

7.1 Introduction

7.1.1 System Overview

CyberDefend is a web application for the general public which can aid them in detecting and preventing common cyberattacks. It mainly consists of different integral features such as detection of spam messages (with or without URLs), discovering past visits to malicious websites, safe search, and detecting dangerous IP addresses. The application takes input data from the user in the form of text, URLs, JSON, and IP addresses and classifies the data as malicious or benign. It's a software program, designed to tackle the rising challenges in cybersecurity faced by the netizens.

The Software Application is divided into 4 modules which need to be tested individually including the following:

- Spam Message Classifier
- Past Visits Analyser

- Safe Search Classifier
- IP Address Classifier

7.1.2 Test Approach

7.1.2.1 Testing Methods

Black-box testing as shown in fig 7.1 validates the functionality of the software system without diving into the internal structures of the software system. It can be used at every level of software testing: unit, integration, system, and acceptance. We are mainly using this type of testing to make sure that the application created is usable for the end client without any technical glitches.

Figure 7.1: Black Box Testing

Other Types of Testing which will be performed by us include:

- **Unit Testing:** Individual Modules are tested to check their functionality is being performed appropriately. This is done in parallel with the development phase.
- **Compatibility Testing:** Compatibility testing to check whether the software is capable of running on different operating systems, web browsers, or Mobile devices.
- **Usability Testing:** The Data must be correct. The user interface should be easily accessible and the content must be correct.

- **Acceptance Testing** Alpha testing is performed to identify software bugs before releasing the software to the end-users. The process involves simulating real users by using black box and white box testing techniques. The goal here is to carry out the tasks that a typical user might perform. It is carried out in a lab environment and the testers are internal to the organization.

7.2 Test Plan

The objectives supported by the Test Plan are:

- To test the project such that it meets the business and user requirements.
- The Test Plan should find defects that may get created by the programmer while developing the software
- To make sure that the result is correct.
- Testing can help gain confidence in and provide information about the level of quality.

7.2.1 Features to be Tested

Software modules that need to be tested are as follows:

- Spam Message Classifier
- Past Visits Analyser
- Safe Search
- Dangerous IP address detector

7.2.2 Features not to be Tested

Features that need not be tested are as follows:

- Performance Issues
- Response Time of APIs

7.2.3 Testing Tools and Environment

As soon as the development is completed for individual modules, the testers are to be ready with the test cases and start with its execution. A manual approach for creating the software will be applied. Each Module is developed separately using HTML, CSS, Bootstrap, Python, and Streamlit as its developing language and framework along with additional APIs and requirements as and when required. This developed application must be integrated using version control software. For manual testing, Tester creates different environments and solves the issues using the various manual as well as automated tools available for Python and Streamlit.

7.3 Test Cases

The Test Cases are shown in fig 7.2 to 7.4

Sr No.	Test Case Name	Test Case ID	Precondition	Description	Input	Expected Output	Actual Output	Status
1	Spam Classifier	1.1	The user is on the Spam Message Detection page	To Verify Sanitization of Text is working properly to prevent Injection	Text Containing JavaScript	Message is displayed without script tags	Message is displayed without script tags	Pass
		1.2	The user is on the Spam Message Detection page.	To Verify Analyze Button working Properly	Analyze Button clicked with Valid Text Input in the text box	Analyze spinner starts spinning	Analyze spinner starts spinning	Pass
		1.3	Analyze Button clicked.	To Verify the output format for text with and without URL	A valid Text message with a URL present	Text message along with class is displayed and URLs are tabulated with class details	Text message along with class is displayed and URLs are tabulated with class details	Pass
					A valid Text message with no URL present	Only text class displayed with no table displayed	Only text class displayed with no table displayed	Pass
		1.4	Analyze Button clicked.	To Verify Blank text input is not accepted	Blank Text-Box	Error message: Please enter the message first!	Error message:Please enter the message first!	Pass
		1.5	Message entered and analyze button clicked	To verify if appropriate messages are displayed.	Spam message	Message: Message is spam!	Displayed Message: Message is not spam!	Fail
		1.6	Message entered and analyze button clicked	To verify if appropriate messages are displayed.	Non Spam message	Message: Message is not spam!	Displayed Message: Message is not spam!	Pass

Figure 7.2: Spam Message Classifier Test Cases

Sr No.	Test Case Name	Test Case ID	Precondition	Description	Input	Expected Output	Actual Output	Status
2	Past Visits	2.1	User Clicks the upload button	Verify if the user is allowed to upload only JSON format of history	CSV file containing the previous URL VIsits	Error with appropriate message.	Error with appropriate message.	Pass
					JSON file containing the previous URL VIsits	Upload successful	Upload successful	Pass
		2.2	User Clicks Analyze button	Verify that a request to classify past visits is not sent before uploading a file in the valid format	No history file uploaded	Error: Please upload a JSON file.	Error: Please upload a JSON file.	Pass
					Valid history file uploaded in JSON format	Analyze spinner starts spinning	Analyze spinner starts spinning	Pass
		2.3	Analysis of the JSON file complete	Verify if the appropriate output is displayed in case of valid input.	URL Classifier Classifies all the URLs in the uploaded history as "Safe"	Your Browsing history seems to be Safe	Your Browsing history seems to be Safe	Pass
					URL Classifier Classifies one or more URLs in the uploaded history as either spam/phishing/ malware.	Message : You have potentially visited malicious pages! Display the title, URL and Class in a tabular format	Message : You have potentially visited malicious pages! Title, URL and Class is displayed in a tabular format	Pass

Figure 7.3: Past Visits Test Cases

Sr No.	Test Case Name	Test Case ID	Precondition	Description	Input	Expected Output	Actual Output	Status
3	Safe Search	3.1	The user lands on the Safe search page	Verify if Sanitization of Text is done to prevent Injection of malicious code	Text Containing JavaScript Code	Error message: Please enter a valid URL.	Error message: Please enter a valid URL.	Pass
		3.2	The user lands on the Safe search page	Verify if analyze button working Properly	Analyze Button clicked with Valid Text Input	Analyze spinner starts spinning	Analyze spinner starts spinning	Pass
		3.3	The user lands on the Safe search page	Verify if blank input is not accepted.	Analyze Button clicked with blank Input	Error message: Please enter a URL first.	Error message: Please enter a URL first.	Pass
		3.4	The user lands on the Safe search page	Verify if an invalid URL is entered, an error message is displayed.	Analyze Button clicked with invalid URL as input.	Error message: Please enter a valid URL.	Error message: Please enter a valid URL.	Pass
		3.5	The user lands on the Safe search page and enters a valid URL	Verify if the appropriate output is displayed in case of valid input.	Analyze Button clicked with valid URL as an input, classified as safe.	Class of URL is displayed and a link to the safe URL is provided.	Class of URL is displayed and a link to the safe URL is provided.	Pass
					Analyze Button clicked with a valid URL as an input, classified as anything except safe.	The class of the URL is displayed correctly. No link should be displayed.	The class of the URL is displayed correctly. No link is displayed.	Pass

		3.6	The user lands on the Safe search page and enters a valid URL	Verify if the appropriate message is displayed.	Phishing URL	Message : URL may potentially lead to deceptive/ phishing website!	Message : URL is likely to be safe!	Fail
					Malicious URL	Message: URL may potentially infect your device with malware!	Message: URL may potentially infect your device with malware!	Pass
					Spam URL	Message: URL may potentially be spam!	Message: URL may potentially be spam!	Pass
					Safe URL	Message: URL is likely to be safe!	Message: URL is likely to be safe!	Pass

Figure 7.4: Safe Search Classifier Test Cases

7.4 Compatibility Testing

Compatibility testing done is shown in fig 7.5 and 7.6 for the web application hosted on Streamlit cloud. The application runs smoothly on Chrome, Firefox, and Edge browsers over Windows 10, Windows 11, and MacOS Catalina operating systems. Given below are screenshots of the tests done using SauceLabs, an online tool for cross-browser testing.

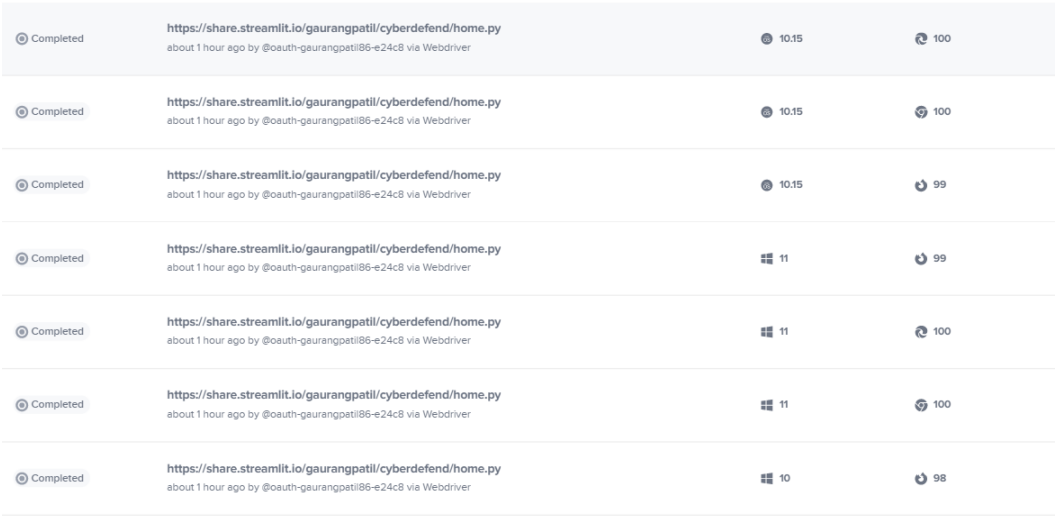


Figure 7.5: Compatibility testing Screenshot 1

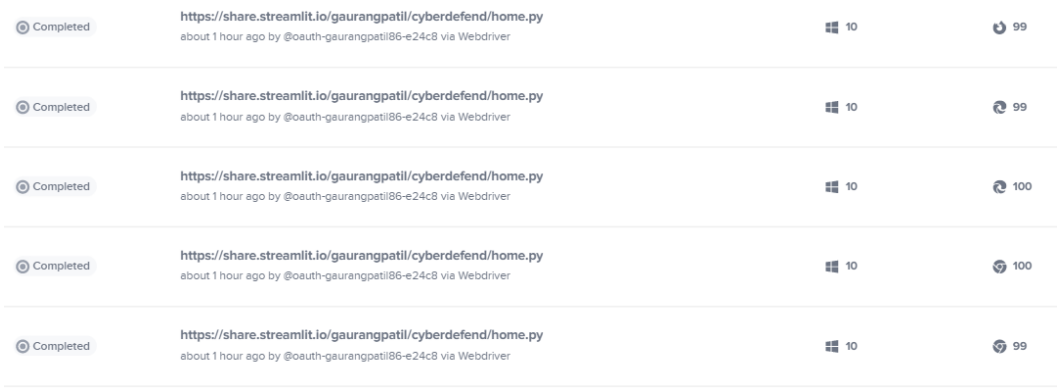


Figure 7.6: Compatibility testing Screenshot 2

7.5 Usability Testing

The performance metrics obtained as the result of Usability testing are summarised in table 7.1. The overall performance score of the web application hosted on Streamlit cloud as generated by GTmetrix is 81%. While most of the important metrics have been adhered to, the cumulative layout shift (as the page loads) needs improvement. The speed index needs slight improvement as well. Given below is a summary of the usability testing report generated by GTmetrix.

Metric	Value	Comments
First Contentful Paint	613 ms	Good user experience
Time to interactive	613 ms	Good user experience
Speed index	2.2 seconds	Greater than recommended values
Total blocking time	0 ms	Good user experience
Largest Contentful Paint	613 ms	Good user experience
Cumulative Layout shift	0.68	Needs improvement

Table 7.1: Performance Metrics

Summary: The chapter elaborated on the unit testing, compatibility testing and usability testing details of the web application. The next chapter discusses the results of the research done as a part of the project.

CHAPTER 8

Results and Discussion

8.1 Results For URL Classifier

The URL classifier uses lexical, JS-based, and HTML-based features to classify the URL into one of the 4 types- Safe, Spam, Phishing, or Malware. Table 8.1 tabulates the 24 features which are used for classification.

Has IP address	No. of Digits	iframe present	Digit To Letter Ratio
Has '@' Sign	No. of Sub Domains	Mouse Over	No. of special characters
Prefix-Suffix	URL Shannon entropy	Has possibly malicious extensions	No. of suspicious words
No. of Dots	Domain shannon entropy	Has .htm extension	No. of image count
Tiny URL	Path Shannon Entropy	Length of URL	Has .html extension
Query path shannon entropy	Query shannon entropy	No. of slash	Has .php extension

Table 8.1: Features Used in URL classifier

Results of the URL classifier are tabulated in Table 8.2. The Random Forest model performs slightly better than XGBoost in classifying the URLs with an overall accuracy of 0.9469. The feature engineering of suspicious file extensions creates 4 new features, as given below:

- HasMalwExt: This feature is set to 1 if a file extension among these - .m, .a, .i, .sh, .exe is present in the URL, else set to 0.
- HasHTMext: This feature is set to 1 if a file extension '.htm' is present in the URL, else set to 0.
- HasHTMLext: This feature is set to 1 if a file extension '.html' is present in the URL, else set to 0.
- HasPHPext: This feature is set to 1 if a file extension '.php' is present in the URL, else set to 0.

ML Model	**Testing Accuracy with features taken from different sources**	**Testing Accuracy after feature engineering**
Random Forest	0.9371	0.9469
XGBoost	0.9353	0.9445
Adaboost	0.7888	0.8168

Table 8.2: Results for URL classifier

Experiments show that these newly engineered features help in increasing the overall performance of the models. Fig 8.1 shows the confusion matrix for the Random Forest classifier and Table 8.3 depicts the categorical accuracy for each category of URL.

Category	Categorical Accuracy
Safe	92.26%
Spam	89.5%
Phishing	99.63%
Malware	98.56%

Table 8.3: Results for URL classifier

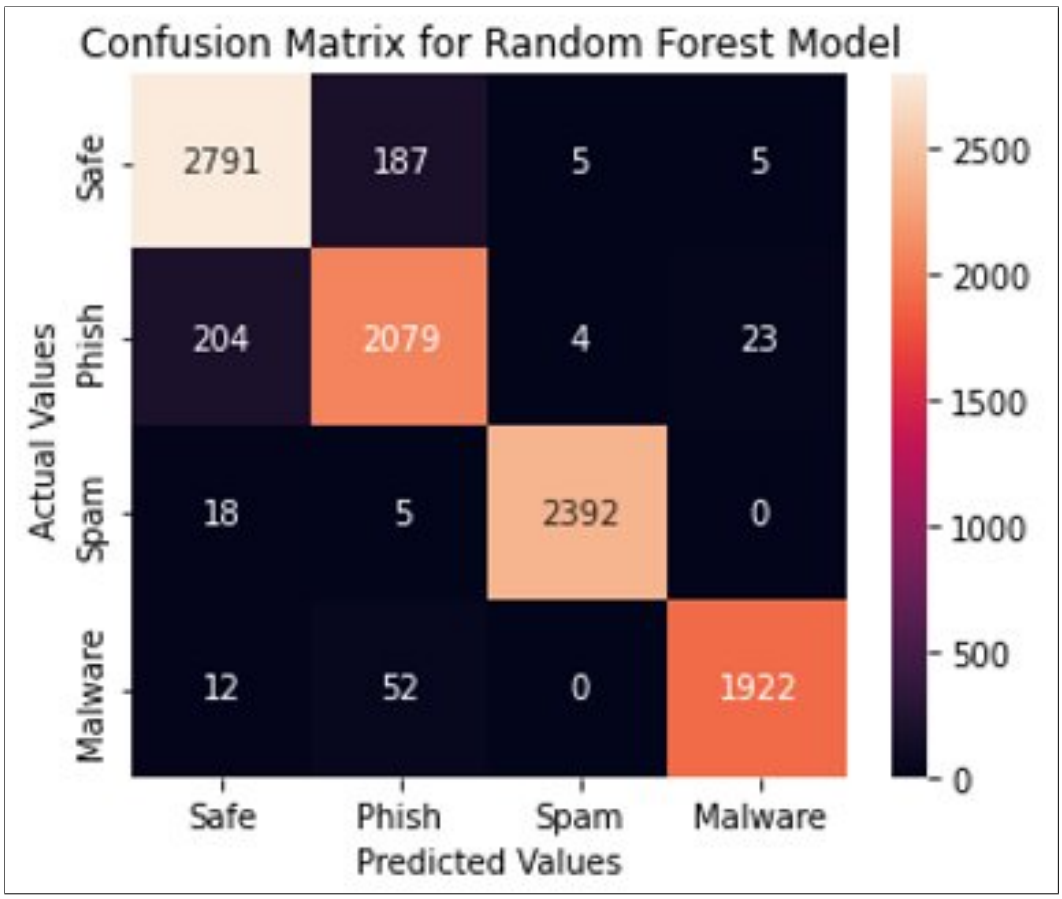


Figure 8.1: Confusion Matrix for Random Forest Classifier

8.2 Results For Spam Classifier

The performance metrics of the machine learning model, 1-channeled, 2-channeled, and 3-channeled models are tabulated in Tables 8.4,8.6,8.7, and 8.8 respectively. SVM has the highest accuracy (77.28%) among the ML models with an F1 score of 0.7008. Naive Bayes has a high recall value but the F1 score is very low, indicating that the model has a high bias. Decision Tree overfits the dataset and is unable to generalize well on the testing dataset.

		Logistic Regression	**Decision Tree**	**SVM**	**Naive Bayes**
Train	Accuracy	0.6757	1.0000	0.8373	0.3607
Test	Accuracy	0.6941	0.7477	0.7728	0.3575
	Recall	0.6941	0.6447	0.7664	0.9770
	F1	0.5315	0.6395	0.7008	0.5147

Table 8.4: Performance Metric of ML Models

Deep learning models give better results when compared to ML models. M3P model with an accuracy of 96.12% and F1 score of 0.9469 outperforms all other variant models. It uses static pre-trained and non-static random embeddings which enhance the representation of the input and helps learn better and reinforced text pattern detecting filters. This leads to better generalization and thus helps obtain a good accuracy on testing data.

	Number of Channels		
	1	**2**	**3**
Convolution Layers arranged in Parallel	M1P	M2P	M3P
Convolution Layers arranged in Series	M1S	M2S	M3S

Table 8.5: Variant Model Notations

		M1S			M1P		
		G	**R**	**W**	**G**	**R**	**W**
Train	Accuracy	0.9630	0.9943	0.9389	0.9636	0.9945	0.9340
Test	Accuracy	0.9053	0.9509	0.9224	0.9189	0.9566	0.9167
	Recall	0.9111	0.8922	0.9017	0.9179	0.8938	0.8973
	F1	0.8557	0.9327	0.8867	0.8786	0.9410	0.8777

Table 8.6: Performance Metric of Models with 1 Channel G: GloVe; R: Random; W: Word2Vec

		M2S			M2P		
		G+W	**G+R**	**R+W**	**G+W**	**G+R**	**R+W**
Train	Accuracy	0.9670	0.9947	0.9935	0.9722	0.9949	0.9942
Test	Accuracy	0.9121	0.9269	0.9532	0.9098	0.9532	0.9463
	Recall	0.8851	0.8320	0.9000	0.8576	0.8860	0.8750
	F1	0.8719	0.9042	0.9354	0.8728	0.9366	0.9276

Table 8.7: Performance Metric of Models with 2 Channel G: GloVe; R: Random; W: Word2Vec

		M3S	M3P
		G+R+W	**G+R+W**
Train	Accuracy	0.9924	0.9957
Test	Accuracy	0.9566	0.9612
	Recall	0.8938	0.9045
	F1	0.9410	0.9469

Table 8.8: Performance Metric of Models with 3 Channel G: GloVe; R: Random; W: Word2Vec

Summary: We use the random forest model for URL classification and the M3P model for spam message classification. The next chapter elaborates on the conclusions drawn from the project and the future scope.

CHAPTER 9

Conclusion and Future Work

The final chapter concludes the thesis with a detailed conclusion and explains the future scope of the project undertaken by the team.

9.1 Conclusion

CyberDefend is an innovative idea that provides protection against cyber-attacks by employing machine learning techniques. The opportunities provided by this e-medium are immense and the general public looking to recognize and prevent common cyber threats can hop on to our web application and avail features such as safe search and spam messages, discovering past visits to malicious websites, browsing safely to prevent visits to malicious/spam/phishing web pages. In today's society, it is certainly vital to employ machine learning approaches in cybersecurity as it aids the netizens to recognize and prevent cyber attacks. To conclude, the objective of designing this system is to lend a helping hand to the users aiming to identify and avert malicious cyberattacks. By using our system, one will be able to combat these attacks smartly.

9.2 Scope for Future Work

In the future our application could be further developed to accommodate the following feature:

1. Directly Report Suspected malicious links to platforms like Phishtank, URLhaus.
2. Implement the Malicious IP address detection functionality.
3. Integrate the platform in which popular messaging and social media services increase the trust factor of information and links received from these platforms.
4. Models can be made more robust by using additional data and can be tested in production, rather than just experimental setups.
5. The machine learning and deep learning models can be integrated with mobile applications such that the messages received by the user can be directly flagged, without the explicit involvement of the user.
6. Similar tool can be built to automatically detect malicious URLs in emails received by the user.

Summary: The tool is useful in detecting social engineering attacks and spam messages. The machine learning and deep learning models can be integrated with mobile applications such that the messages received by the user can be flagged.

CHAPTER 10

References

[1] T. Manyumwa, P. F. Chapita, H. Wu and S. Ji, *Towards Fighting Cybercrime: Malicious URL Attack Type Detection using Multiclass Classification*; 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 1813-1822, doi: 10.1109/BigData50022.2020.9378029.

[2] Rishikesh Mahajan and Irfan Siddavatam, *Phishing Website Detection using Machine Learning Algorithms*; International Journal of Computer Applications 181(23):45-47, October 2018

[3] J. Fattahi and M. Mejri, *SpaML: a Bimodal Ensemble Learning Spam Detector based on NLP Techniques,*; 2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP), 2021, pp. 107-112, doi: 10.1109/CSP51677.2021.9357595.

[4] M. RAZA, N. D. Jayasinghe and M. M. A. Muslam, *A Comprehensive Review on Email Spam Classification using Machine Learning Algorithms,* 2021 International Conference on Information Networking (ICOIN), 2021, pp. 327-332, doi: 10.1109/ICOIN50884.2021.9334020.

[5] Y. -C. Chen, Y. -W. Ma and J. -L. Chen, *Intelligent Malicious URL Detection with Feature Analysis,* 2020 IEEE Symposium on Computers and Communications (ISCC), 2020, pp. 1-5, doi: 10.1109/ISCC50000.

2020.9219637.

[6] M. A. Parwez, M. Abulaish and Jahiruddin, *Multi-Label Classification of Microblogging Texts Using Convolution Neural Network,* in IEEE Access, vol. 7, pp. 68678-68691, 2019,
doi: 10.1109/ACCESS.2019.2919494.

[7] "Don't let hackers make tax season even worse.," Webroot. [Online]. Available: https://mypage.webroot.com/threatreport.html. [Accessed: 15-Jul-2021].

[8] PTI, "3.17 Lakh cybercrimes in India in just 18 months, says govt," Return to frontpage, 09-Mar-2021. [Online].
Available: https://www.thehindu.com/sci-tech/technology/317-lakhs-cybercrimes-in-india-in-just-18-months-says-govt/article34027225.ece [Accessed: 15-May-2021].

[9] Bureau, Our. "Phishing Remains the Top Threat to Consumer Safety: Report." @Businessline, The Hindu BusinessLine, 14 June 2021, https://www.thehindubusinessline.com/info-tech/phishing-remains-the-top-threat-to-consumer-safety-report/ article34810027.ece.

[10] "Aussies Losing Less Money to Scams but SMS Phishing on the Rise." EFTM, 20 May 2021,
https://eftm.com/2021/05/aussies-losing-less-money-to-scams-but-sms-phishing-on-the-rise-211207.

[11] "What is social engineering? (expert explains attacks prevention),"

PurpleSec, 13-Sep-2020. [Online]. Available: https://purplesec.us/social-engineering/. [Accessed: 15-Apr-2022].

[12] Social-Engineer, M. R. | A. 07, M. A. | F. 17, B. S. | F. 03, Teri Robinson | 4 hours ago, and M. V. | Yesterday, “Social Engineering in the news: Smishing,” Security Boulevard, 19-Jan-2022. [Online]. Available: https://securityboulevard.com/2022/01/social-engineering-in-the-news-smishing/. [Accessed: 15-Apr-2022].

CHAPTER 11

Acknowledgment

The successful completion of this project would not have been possible without the help and guidance of many respected individuals. It gives us great pleasure to express our sincere thanks and gratitude to all of them. We are extremely grateful to our project guide **Dr. Sonali Patil** for her constant support and encouragement throughout the course of this project.

In addition, we are extremely thankful to our respected Principal, and the Head of IT Department, **Dr. Ujwala Bhangale**, for giving us the support and guidance to work on this project. Our thanks and appreciation go to the college and the staff for providing us with the necessary resources and valuable suggestions. We express our sincere heartfelt gratitude to each individual who helped us successfully develop our project within the prescribed time.

Date:

Spam Classification Leveraging Multi-Channel Convolution Neural Network

Gopalkrishna Waja[a,*], Gaurang Patil[a], Charmee Mehta[a] and Dr. Sonali Patil[a]

[a]K.J. Somaiya College of Engineering, Vidya Vihar, Ghatkopar East, Mumbai, 400077, India

ARTICLE INFO

Keywords:
multi-channel
convolution neural network
spam classification
word embeddings
natural language processing
SMS text messages

ABSTRACT

Over the past decade, we have seen a meteoric evolution in the Internet Messing Services and although these services have become engrained in our everyday life, SMS service remains, indisputably, an essential form of communication service. SMS remains one of the most widely used communication services with over 5 billion users globally sending and receiving SMS messages. The omnipresence of SMS has also given rise to unsolicited and junk messages also known as SMS Spam which has motivated researchers to use machine learning and deep learning to detect and control these kinds of messages. Machine learning techniques generally represent the text in form of bag of words or n-gram feature vectors but since the SMSs generally have very few words, the vectors produced are sparse and suffer from the curse of dimensionality. Recently, studies using deep learning have shown promising results for spam classification, and in this paper, extending these studies, we have proposed a Multi-Channel Convolutional Neural Network (CNN) architectures with static and dynamic embeddings for SMS spam classification. To evaluate the performance of the proposed model we have compared the proposed model's performance metrics with that of various variant models created by altering the number of channels and orientation of convolution layers and some popular machine learning models.

1. Introduction

Short Message Service abbreviated as SMS is considered to be one of the quickest and most economical forms of communication. SMS has entrenched itself as both a formal and informal medium of communicating. Every day people receive a plethora of messages, some of which are solicited like messages from friends and acquaintances, bank transaction notifications, OTPs, and messages from other services, while others are unsolicited like promotional and phishing messages which are also known as spam messages. Sometimes these spam messages, although annoying, appear harmless but often they provide a medium to the attackers for stealing users' private details, like card numbers, passwords, bank account details, etc, and hence spam message detection has become a necessity.

SMS spam classification is a problem that is still a subject of research for which lots of efforts have been made to present a simple yet robust solution. Many studies in the past have used machine learning techniques for text classification which although have given quite successful performance, have faced challenges like manual feature extraction by domain experts, difficulty in preserving word order and context, overfitting, and scalability Minaee, Kalchbrenner, Cambria, Nikzad, Chenaghlu and Gao (2021). For this reason, researchers have started to shift their attention towards deep learning approaches to obtain better performance, semantic and syntactic disambiguation, and scalability by the use of word vector representation and automated feature extraction.

Special networks are trained in a variety of Natural Language Processing (NLP) tasks to get words or sentence representations are learned as vectors that encode the semantic and syntactic information about the input which are then used for automated extraction of higher-level features required for classification Parwez, Abulaish and Jahiruddin (2019). Over the recent years, Convolutional Neural Network (CNN) models have portrayed tremendous potential for extraction of these high-level features and hence aid in text classification. In this paper, we have proposed a Multi-Channel CNN architecture with a combination of static pre-trained word embeddings and non-static random word embeddings for solving the spam classification problem Kim (2014). For pre-trained embeddings, we have chosen GloVe and Word2Vec. The proposed model provides twofold benefits. Firstly, it uses multiple channels of word embeddings instead of a single or none which provides a varied non-sparse representation of the input required for better capturing the semantic information. Secondly, orienting the convolutions layers in parallel enable them to act as automatic n-gram feature detectors for each channel Jacovi, Shalom and Goldberg (2020). Thus we feel that using this model we can overcome the limitations posed by other machine learning techniques. The model is trained on a dataset created by merging UCI's spam dataset and self-collected messages. To gauge the model's effectiveness we have additionally trained and compared other common machine learning algorithms and variants of the proposed model by changing the number of channels and orientation.

The organization of the upcoming sections is as follows. The section2 provides a brief review of the existing spam classification techniques. Following this section 3 provides details about the dataset and the proposed model. Then in section 4 and 5 we have described the experimental setup,

This document is the result of the LY B.tech project under K.J. Somaiya College of Engineering.

*Corresponding author

gopalkrishna.w@somaiya.edu (G. Waja); gaurang.patil@somaiya.edu (G. Patil); charmee.m@somaiya.edu (C. Mehta); sonalipatil@somaiya.edu (S. Patil)

ORCID(s):

results, and conclusion and we have ended this paper by providing a conclusion in section6.

2. Related Work

The Spam classification problem, over the past decade, has been studied widely by researchers over the world. The work done by them over these years can be broadly categorized into three approaches, namely rule-based techniques, machine learning techniques, and deep learning techniques.

Rule-based approaches are based on using pre-defined manually designed language rules, developed by domain experts, in form of regular expressions which identify characteristic phrases to distinguish spam from non-spam messages. So generally, in these systems, the corpus is scanned for these rules, and if matches are discovered, their weight is added to the overall score which is used for final classification Kaddoura, Chandrasekaran, Popescu and Duraisamy (2022). Luo, Liu, Yan and He (2011) Luo Q has used rule-based extraction and filtering with dynamic adjustment of static rules on the Spam Assassin corpus containing 4,150 spam and 1,897 ham emails to optimize the spam filtering process. Using this approach gave them an accuracy of 98.5%, a false-positive rate of 0.42%, and a false negative rate of 4.7%. Shrivastava and Bindu (2014) Shrivastava and Bindu have used a rule-based spam detection filter with pre-assigned weights in combination with Genetic Algorithm for efficient spam detection which gave them an accuracy of 82.7% and a precision of 83.5%. Similarly Saidani, Adi and Allili (2020), Saidani and Adi have utilized manually and automatically extracted rules from the Enron dataset having 2,893 with domain categorization to obtain accuracy and precision of 98.0% and 0.98 respectively. Although we can see that rule-based approaches provide good performance, they primarily depend on static rules and therefore cannot be generalized well to the ever-changing spam content, furthermore, it might not be always possible to identify all the rules required for spam detection. In addition to this, although automated rule generation systems are available to constantly modify rules based on changing spam content, their limitation is that they require a significantly larger amount of context-based features which increases time, analysis, and computational complexity. All these limitations associated with the rule-based approach have served as a motivation for the adoption of machine learning and deep learning for spam message classification.

In the case of machine learning, researchers have studied the effect of using some of the most popular algorithms like the naive Bayes, decision trees, random forest, support vector machines, and ensemble learning for spam classification. In Mishra and Soni (2020), the authors had proposed a model named "Smishing Detector" containing different modules to analyze different parts of the message from UCI's SMS spam dataset. In this paper, they had used the naive Bayes classifier to obtain a precision and accuracy of 91.6% and 93%. In another paper Sonowal and Kuppusamy (2018), Sonowal has proposed a model, called "SmiDCA", for the detection of SMS spam messages from the UCI SMS spam dataset in which he uses correlation analysis for feature extraction followed by the application of 4 machine learning algorithms, random forest, decision tree, support vector machine, and AdaBoost to perform a comparative analysis. Here, the best accuracy of 96.4% was obtained for the random forest model. In Fattahi and Mejri (2021), authors have proposed "SpaML" model in which feature extraction was done using a Bag of words and TF-IDF, and classification was performed using a majority of the results of an ensemble of 7 models i.e. multinomial naive Bayes, SVM, nearest centroid center, logistic regression, Xgboost, KNN, and perceptron. This model was trained on UCI's Spam dataset and gave an accuracy of 97%. In Jain and Gupta (2019), the authors propose ten features that they claim to be able to distinguish false messages from ham. After this, they test the effectiveness of these features by training and comparing random forest, naive Bayes, SVM and Neural Network. This indicated that random forest gave the highest true positive rate of 94.20% and 98.74% of overall accuracy.

Having studied the prior work done by researchers on spam classification using machine learning we can understand that these techniques provide us the ability to deal with dynamic conditions and help to counter the limitations posed by rule-based systems. Despite this, they have their own set of limitations. In general they either extract features manually using domain knowledge or use techniques like bag of words to provide a representation which is often sparse. In particular, Algorithms like support vector machines that analyze and identify patterns in data for classification, though perform well on a small dataset, fail to generalize well and become computationally impractical on larger datasets Kaddoura et al. (2022). Other algorithms which make up tree structures like decision trees suffer from difficulties like controlling tree growth and over-fitting of training data. In probabilistic models like naive Bayes classifier makes a simplistic assumption that each word in a message is independent of others which is rarely the case in real-life SMS messages. Finally, studies indicate that ensemble algorithms frequently are cursed by high computational complexity and domain dependence. To avoid such deterrents by ML techniques, researchers have lately focused their attention on deep learning techniques.

Deep learning models have taken the world by storm due to their applicability to myriad domains, including natural language processing, the ability to deal with the scalability issue and extract the features automatically Kaddoura et al. (2022). The most prominent deep learning models among NLP researchers are Recurrent Neural networks, Long Short Tern Memory (LSTM) and Convolutional Neural Networks (CNN) networks. In Kim (2014), Yoon Kim had proposed different types of CNN model variants for text classification in general which over the years have formed the benchmark for using CNN for natural language data. In Roy, Singh and Banerjee (2020), authors have used single-channel CNN in combination with LSTM on the UCI's spam dataset and obtained an accuracy of 99.4%. Popovac et al Popovac,

Karanovic, Sladojevic, Arsenovic and Anderla (2018) have proposed a CNN-based architecture having a single convolution and pooling layer SMS spam classification which achieved an accuracy of 98.4%. In Jain, Sharma and Agarwal (2019), authors used the LSTM network to SMS spam for the same dataset with 200 LSTM nodes and pre-trained word embeddings, to achieve an accuracy of 99.01%. In Yaseen et al. (2021), authors have fine-tuned BERT with Word2Vec on the Open source SpamBase dataset with 5,569 messages to obtain accuracy and F1 score of 98% and 0.98 respectively. In Zhuang, Zhu, Peng and Khurshid (2021), Deep Belief Networks (DBN) are used on WEBSPAM-UK2006 and UK2007 datasets to obtain an accuracy of 94% and a precision of 0.95. Similarly, there are many other studies where researchers have leveraged deep learning techniques for the spam classification problem but to our best knowledge, although they have studied spam classification using single word embedding and CNN, no one has yet explored the effect of using Multi-channel CNN architecture with different orientations of convolution layers for SMS spam classification which we aim to do in this paper.

3. Approach Followed

In this section, we will discuss the approach followed for the development of the spam message classifier. First, we describe the dataset which we have used for the spam classification task, the data sources whose amalgamation dataset is, and the composition of these individual data sources. After this, we elucidate the data pre-processing and augmentation pipeline through which the textual messages pass before they are fed to the spam classification model for training and testing. Finally, we have explained the proposed multi-channel CNN model containing static and non-static embeddings for spam classification.

3.1. Preparing Dataset

The pre-existing open-source datasets often have a limited variety of text messages and hence the dataset which we have used is a combination of a pre-existing dataset and a self-created dataset formed by labeling messages collected from our mobile phones over 6 months. The self-collected messages include a variety of banking, e-commerce, and other types of messages.

Table 1
Dataset Description

Dataset/Type	Spam	Non-Spam	*Total*
UCI Dataset	653	4516	5169
Self-Collected	2429	1153	3582
Total	3082	5669	8751

The pre-existing dataset is UCI's public dataset containing 5169 unique labeled SMSs out of which, as visible in fig.1, 653 are spam and 4516 are non-spam. In addition to this our self-collected dataset contains 3582 unique messages with 2429 spam and 1153 non-spam messages. So, cumulatively the final dataset contains 8751 messages out of which 3082 are spam and 5669 are non-spam.

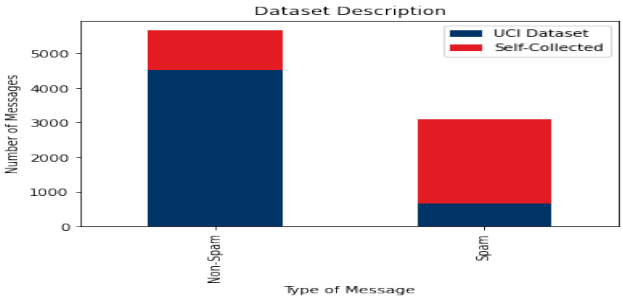


Figure 1: Stacked bar plot for dataset label count

3.2. Data Augmentation and Processing

Pre-processing and cleaning the data involves performing operations on the raw text data to transform it in a proper format so that meaningful information can be extracted from it. In our case pre-processing involved the removal of leading and trailing blank spaces, extraction of URLs, and replacement of digits and URLs with blank spaces. Following this data augmentation is performed on the training messages. Text data augmentation helps to improve the generalizability of the model by increasing the number and variety of training examples which in turn prevents the model from overfitting the data. For this paper, text data augmentation is done by duplication and producing 2 synthetic sentences, by replacement of words by their synonyms, for each sentence in the training set facilitating an augmentation from 7875 training messages to 15750 messages. Once augmentation is done tokenization followed by stop-word removal and lemmatization is performed. Tokenization is the process of decomposing a sentence into individual characters, words, or n-grams. For our study, we have used word-tokenization using white-space as delimiters. In stopword removal, we discard the most commonly used words in a language or corpus which generally do not provide any meaningful information required for the classification. Following this, in lemmatization, we aim to convert inflectional forms to a root word. An SMS may contain different inflections like running, runs, ran, etc of the same base word "run", lemmatization uses dictionary and morphological analysis on these words, to eliminate inflections to obtain the root word.

Finally, once the lemmatization of the sentence is done, the sequence lemmas are mapped to a sequence of numbers unique to each lemma to obtain an N-vector representing the processed text.

3.3. Model Architecture

In this section, we will describe the multi-channel CNN model architecture which we have used for spam message classification. The different vector operations performed are

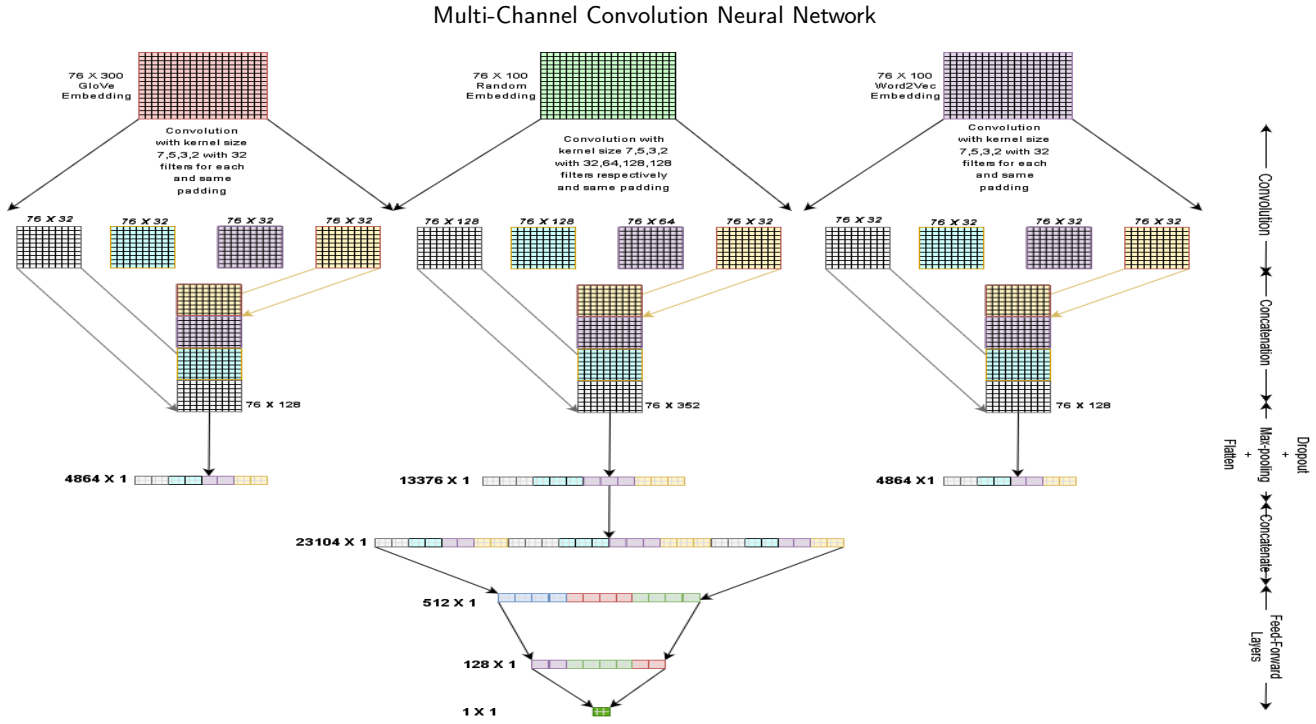


Figure 2: CNN Model Vector Representation

shown in fig.2 and the overall architecture implemented along with layers of different CNN models is shown in fig.3 . It can be seen that the model consists of three channels, corresponding to Word2Vec, Random, and GloVe embeddings respectively. The model takes a sequence of numeric tokens as input and uses trainable embeddings and high-level features extracted through convolution to classify the spam messages. An important feature of this architecture is that it uses a combination of pre-trained and dynamically generated word embeddings which are fine-tuned during training to make it more specific to the task-at-hand and provide a more meaningful representation of the tokens encountered in the spam messages. In addition to this, the model also arranges the convolution layers in parallel instead of series. Specific details regarding each component of the model are as follows:-

3.3.1. Input Layer

The model takes a N-vector as the input which contains a sequence numeric tokens obtained after processing and padding the text data. For our model N is equal to 76 and therefore the input layer accepts a 76-vector representing a text message as the input as depicted in the equation (1).

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_N \end{bmatrix} \tag{1}$$

where $x_1, x_2, ...x_N$ are numeric tokens.

3.3.2. Embedding Layer

The embedding layer maps input vector $\mathbf{X} \in \mathbb{Z}^N$ containing sequences of tokens to a word embedding matrix $\mathbf{E} \in \mathbb{R}^{N,K}$ where $\mathbf{K}$ is the length of each word embedding and each row of the $\mathbf{E}$ is a word vector corresponding to the token in the input vector. Word embedding matrix provides a representation that is fed as input to the upcoming convolution layers which learn high-level features useful to determine the class of these messages. We have used the concept of multi-channel embedding proposed Kim (2014), to obtain better word representation and improved performance by enabling better sematic and syntactic disambiguation. In our model, each input is mapped to 3 embeddings namely to GloVe, Random, and Word2Vec embeddings corresponding to the 3 channels respectively.

$$\mathbf{E_i} = \begin{bmatrix} e_{1,1} & e_{1,2} & . & . & e_{1,Ki} \\ e_{2,1} & e_{2,2} & . & . & e_{2,Ki} \\ . & . & . & . & . \\ . & . & . & . & . \\ e_{N,1} & e_{N,2} & . & . & e_{N,Ki} \end{bmatrix} \tag{2}$$

where,

i is the channel corresponding to the embedding

k_i is the dimension corresponding to the channel

Each row of E_i i.e $< e_{N,1}, e_{N,2}..e_{N,K_i} >$ is word vector corresponding to x_N .

The Word2Vec embeddings are 300-dimensional embeddings pre-trained on Google News data containing billions of words while GloVe embeddings are 100-dimensional embeddings pre-trained on over millions of Wikipedia pages. Out of these millions of vector representations, the word vectors corresponding to the vocabulary of the text in the

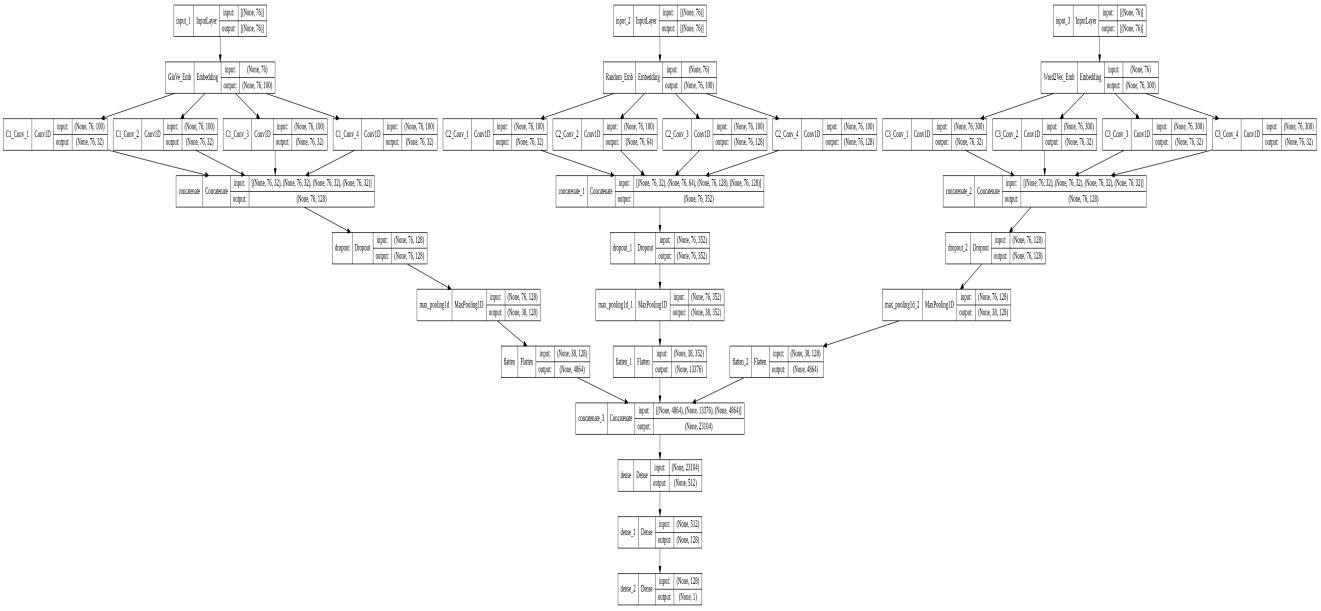


Figure 3: CNN Model Architecture

training set are extracted and used by the model. In case a particular word's GLoVe or Word2vec vector representation is not found, a very common scenario in spam classification due to profuse usage of abbreviations and slang, then in such cases word is by default initialized with a null vector. The same initialization strategy is used for words encountered during testing which are not part of the vocabulary. In addition to this, the random embedding is also a 100-dimensional embedding initialized using a uniform distribution. Furthermore, it should be noted that although the random embedding is a non-static trainable embedding, the GloVe, and Word2Vec embeddings, keeping in mind the small size of the dataset, are kept static to prevent overfitting and also decrease computational complexity.

3.3.3. Convolution Layer and Concatenation

In the case of text classification, we need to perform 1D convolution to extract high-level n-gram patterns required for classification. Convolution filters act as heterogeneous n-gram detectors where each filter is trained to detect one or several families of closely-related n-grams depending on the filter width Jacovi et al. (2020). Thus using multiple convolution layers, with different kernel widths, in parallel to each other in such a manner that they act as a single layer with multiple kernels of varying sizes will allow detection of n-gram phrases of different lengths required for classification. In our model, we have used 4 convolution layers in parallel per channel. Each channel uses convolution layers with kernel size $F_{i,j} \in \mathbb{R}^{w_j,K}$, where $w_j \in 2,3,5,7$ is the width of the kernel of jth layer, and $f_{i,j} \in \mathbb{Z}$ number of filters, where i is the channel number. Over here same convolution operation is performed, meaning the shape of the input remains same after convolution. In-order for this to happen the input matrix E_i must be P-zero padded to obtain $EP_i \in \mathbb{R}^{N+2P,K}$ where P, considering stride as 1, is

$$\mathbf{P_{i,j}} = \frac{F_{i,j} - 1}{2} \tag{3}$$

Once the padding is done the output activation map $A_{i,j,z} \in \mathbb{R}^{N,1}$ where z is the filter number which produces the activation and each individual element a_n of $A_{i,j,z}$ is calaculated as follows:

$$A_{i,j,z} = [a_1, a_2, ..., a_n] \tag{4}$$

$$a_n = g(EP_{n:n+w_j-1} * F_{i,j} + b_n) \tag{5}$$

where,
$EP_{n:n+w_j-1}$ represents using w_j rows at a time
$*$ is the convolution operator
b_n is the biased term
$g(x)$ is the ReLu activation function

Since at each layer there are $f_{i,j}$ filters of same size $f_{i,j}$ such activation are produced which are stacked side-by-side to give a feature map of shape $(N\ ,\ f_{i,j})$. After this, the feature maps of all the 4 convolution layers are concatenated together to obtain C_i having final dimensions as:

$$(N, \sum_{j=1}^{4} f_{i,j}) \tag{6}$$

3.3.4. Dropout Layer

The dropout layer randomly masks out a set of neurons with the specified probability during training which helps to reduce co-adaptation of hidden units and regularize the network to provide better validation score and hence better generalizability.

3.3.5. 1D MaxPooling Layer

This layer helps to reduce the dimensionality of the activation maps and hence reduces the number of parameters and computation needed. 1D MaxPooling Layer takes in the concatenated output of the dropout layer and instead of performing Global Maxpooling, as depicted in Kim (2014) where 1 maximum element from the entire activation was chosen, here a window of size 2 with stride 2 is used for Maxpooling which effectively changes the shape of each $A_{i,j,z}$ in the concatenated output C_i from (N,1) to ($\frac{N-2}{2}+1$, 1) and this is repeated for all the activations giving the final shape of the output as

$$(\frac{N-2}{2}+1, \sum_{j=1}^{4} f_{i,j}) \tag{7}$$

3.3.6. Flatten and Dense Layers

The flatten layer takes the output matrix of the pooling layer and converts it into a vector $D_i \in \mathbb{R}^{(\frac{N-2}{2}+1)\cdot\sum_{j=1}^{4} f_{i,j}}$. Following this, the model concatenates all the flattened layers from the 3 channels on top of one another as shown in (8).

$$\mathbf{D} = D_1 \oplus D_2 \oplus D_3 \tag{8}$$

where $\oplus$ is concatenation operator. Following this D is passed through a fully connected feed-forward network containing two dense layers which preforms the following operation:

$$Y = g(W_2 \cdot g(W_1 \cdot D + b_1) + b_2) \tag{9}$$

where W_1, b_1 and W_2, b_2 are weights and biases corresponding to the 1^{st} and 2^{nd} desnse layer with 512 and 128 neurons respectively. Finally, the output layer has 1 neuron with a sigmoid activation function which outputs the probability of the message being spam.

$$O = Sigmoid(Y) = \frac{1}{1+e^{-Y}} \tag{9}$$

where, $O \in \mathbb{R} \mid 0 \leq O \leq 1$ and the final label l assigned to the input message is determined as

$$l = \begin{cases} 0 & O \leq 0.5 \\ 1 & O > 0.5 \end{cases} \tag{10}$$

where $l = 0$ indicates message is non-spam while $l = 1$ indicates message is spam.

4. Experimental Setup

The model described in the previous section is trained using binary cross-entropy as the loss function, 512 as the batch size, and Adam optimizer which makes use of both first and second-order moments for faster optimization. Table 2 gives the hyper-parameters which are used for training corresponding to the convolution layers in different channels. During training, the dropout regularization value was set to 0.5 and in addition to dropout, early stopping was also used as another regularization technique to prevent overfitting and obtain better validation accuracy. Early Stopping halts the training when some fixed number of successive epochs do not provide an improvement on a validation set and uses the last best parameters which were stored during training.

Table 2
Convolution Layer Hyper-Parameters

Channel number j	Conv Layer, Filter count, Width (j, f_{ij}, w_j)
1	(1, 32, 2) (2, 32, 3) (3, 32, 5) (4, 32, 7)
2	(1, 128, 2) (2, 128, 3) (3, 64, 5) (4, 32, 7)
3	(1, 32, 2) (2, 32, 3) (3, 32, 5) (4, 32, 7)

Table 3
Other Model Hyper-Parameters

Hyper-Parameter	Value
Word embedding size for Glove, Random and Word2vec	100,100 and 300
Dropout probability	0.5
MaxPooling Size	2
Early stopping patience	15
Dense 1 and Dense 2 neurons	512 and 128

Since the model has been trained on a dataset having self-collected messages it is difficult to compare the model's performance against other models trained on different datasets. So to put things into perspective, after training and testing the proposed model, other variants of the CNN model along with some common machine learning models, namely logistic regression, SVM, decision trees, and naive Bayes, have

been trained on the same dataset and their performance is compared against the proposed model. The variant models are created by changing the number of channels and connection of convolution layers in series instead of parallel. When convolution layers are stacked on top of one another in form of a series the first convolution layer acts as the n-gram detector while the subsequent layers detect complex character-level features from the feature map of the first layerZhang, Zhao and LeCun (2015). This type of text convolution is generally helpful when analyzing long sequences of text but in the case of SMSs where the messages are usually short stacking convolution layers should not make much difference in performance and therefore we hypothesize those model variants where the number of channels is the same but the orientation of convolution layers is different will have comparable accuracies. In addition to this, we feel that, since the use of multiple embeddings helps to provide a more diverse representation of the words leading to better disambiguation, models with multiple embeddings will outperform the models with a lower number of embeddings. To verify these notions, we have trained and compared the performance of all these variant models in the next section.

Table 4
Variant Model Notations

	Number of Channels		
	1	**2**	**3**
Convolution Layers arranged in Parallel	M1P	M2P	M3P
Convolution Layers arranged in Series	M1S	M2S	M3S

To simplify the referencing of these models we have used the notation described in the table 4. According to the convention decided, our model which is described in the earlier section with three channels and convolutions in parallel has the notation M3P. Similarly, some other models with 2 channels and convolutions in series will have the notation M2S. In the case of a model with 1 and 2 channels, for each model 3 sub-variants are possible based on the combination of the embeddings used.

For comparing the models we have used accuracy, recall, and F1 score as the performance metric. Accuracy of the model given by

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{11}$$

indicates the fraction amount of the time the classifier is correct, that is When the classifier predicts an SMS is or is not spam, what is the probability that it's correct. Recall given by

$$Recall = \frac{TP}{TP + FN} \tag{12}$$

indicates how many spam messages the model can identify out of all the spam SMSs present in the test set. Finally, the F1 score given by

$$F1 = \frac{2TP}{FP + 2TP + FN} \tag{13}$$

is the harmonic mean of the precision and recall which helps us to compare multiple models having a different recall and precision values.

5. Results and Discussion

In this section, we will compare the results obtained after training and testing the proposed model, the variant models, and the machine learning models considered in the previous section. In addition to this, we will discuss the effect of change in the number of channels on the performance of the model and visualize the activations of the convolution layers of these variant models, by plotting the feature maps, to gain better insights about these models.

The performance metrics obtained by training and testing Machine learning, 3-channeled, 2-channeled, and 1-channeled models are tabulated in the table5,6,78 respectively. It can be seen that ML algorithms give a rather poor performance on the current dataset. The highest accuracy obtained for the ML algorithms is 77.28% with an F1 score of 0.7008 for SVM and the worst performance among ML algorithms is given by naive Bayes with accuracy and F1 score of 36.07% and 0.5147. It can be noticed that in the case of naive Bayes, the recall is high, that is it can identify a large number of spam messages despite this its accuracy and F1 score is very low indicating that the trained model has high biased and is almost every time predicting 1 irrespective of the message and therefore is not able to identify non-spam messages. Another observation that can be made is in the case of the Decision tree algorithm although the training accuracy of 100% is received the accuracy and F1 score on the test set is merely 74.77% and 0.6395, indicating the model is highly over-fitting the training examples and is not able to generalize well to unseen examples.

Table 5
Performance Metric of ML Models

		Logistic Regression	**Decision Tree**	**SVM**	**Naive Bayes**
Train	Accuracy	0.6757	1.0000	0.8373	0.3607
Test	Accuracy	0.6941	0.7477	0.7728	0.3575
	Recall	0.6941	0.6447	0.7664	0.9770
	F1	0.5315	0.6395	0.7008	0.5147

From the table 6,7,8 it can be observed that these deep learning models give far better results than the machine

Table 6
Performance Metric of Models with 1 Channel G:GloVe; R:Random; W:Word2Vec

		M1S			M1P		
		G	R	W	G	R	W
Train	Accuracy	0.9630	0.9943	0.9389	0.9636	0.9945	0.9340
Test	Accuracy	0.9053	0.9509	0.9224	0.9189	0.9566	0.9167
	Recall	0.9111	0.8922	0.9017	0.9179	0.8938	0.8973
	F1	0.8557	0.9327	0.8867	0.8786	0.9410	0.8777

Table 7
Performance Metric of Models with 2 Channel G:GloVe; R:Random; W:Word2Vec

		M2S			M2P		
		G+W	G+R	R+W	G+W	G+R	R+W
Train	Accuracy	0.9670	0.9947	0.9935	0.9722	0.9949	0.9942
Test	Accuracy	0.9121	0.9269	0.9532	0.9098	0.9532	0.9463
	Recall	0.8851	0.8320	0.9000	0.8576	0.8860	0.8750
	F1	0.8719	0.9042	0.9354	0.8728	0.9366	0.9276

learning models and the limitations of high bias and overfitting which were seen in the previous case seem to have been mitigated by these models. Comparing these three tables we can see that our proposed model M3P with an accuracy of 96.12% and F1 score of 0.9469 outperforms all other variant models, albeit in some models variants like M1P with random embedding, M2P with Glove and random, M2P with Word2Vec and random and M3S the difference in performance is not much.

Table 8
Performance Metric of Models with 3 Channel G:GloVe; R:Random; W:Word2Vec

		M3S	M3P
		G+R+W	G+R+W
Train	Accuracy	0.9924	0.9957
Test	Accuracy	0.9566	0.9612
	Recall	0.8938	0.9045
	F1	0.9410	0.9469

It can also be seen that in the case of single channeled models without random embedding some models have recall greater than our proposed M3P model although their accuracy and F1 score is lower than our model. This is can be explained by the fact that single channeled models with static embeddings provide a restricted word representation when compare to models with a higher number of channels leading to some bias and more frequent prediction of 1's than 0's causing higher recall but lower accuracy and F1 score.

It can be observed that single channeled models M1P and M1S have the least overall testing accuracy, M2P and M2S model's accuracy is slightly more than single channeled models but less than the 3 channeled models. There are a few observations that bring out some important points regarding these variant models which helps us to explain why the M3P model outperforms the rest.

Firstly, we can see that in the case of single-channel models, using non-static random embedding gives better performance than using static pre-trained embeddings. Similar observation can be made for a model with two channels where a combination of random and pre-trained embeddings gave better accuracy than using pre-trained embedding in both the channels. Generally, it is seen that using pre-trained embeddings allows efficient learning by capturing the semantic and syntactic meaning of a word and hence facilitating the improved performance of NLP models Goldberg (2017). Therefore, it is a natural instinct to think that pre-trained embeddings might out-perform models with random embeddings, but where this is not the case. This observation can be, according to us, primarily be explained as a consequence of three factors. Firstly, the pre-trained embeddings which are available, that is GloVe and Word2Vec, have been trained on Wikipedia and Google news data which is vastly different from the spam messages data which has an eclectic variety of abbreviations and slang and hence the semantic meaning and synthetic structured represented by the pre-trained embeddings might possibly be slightly different

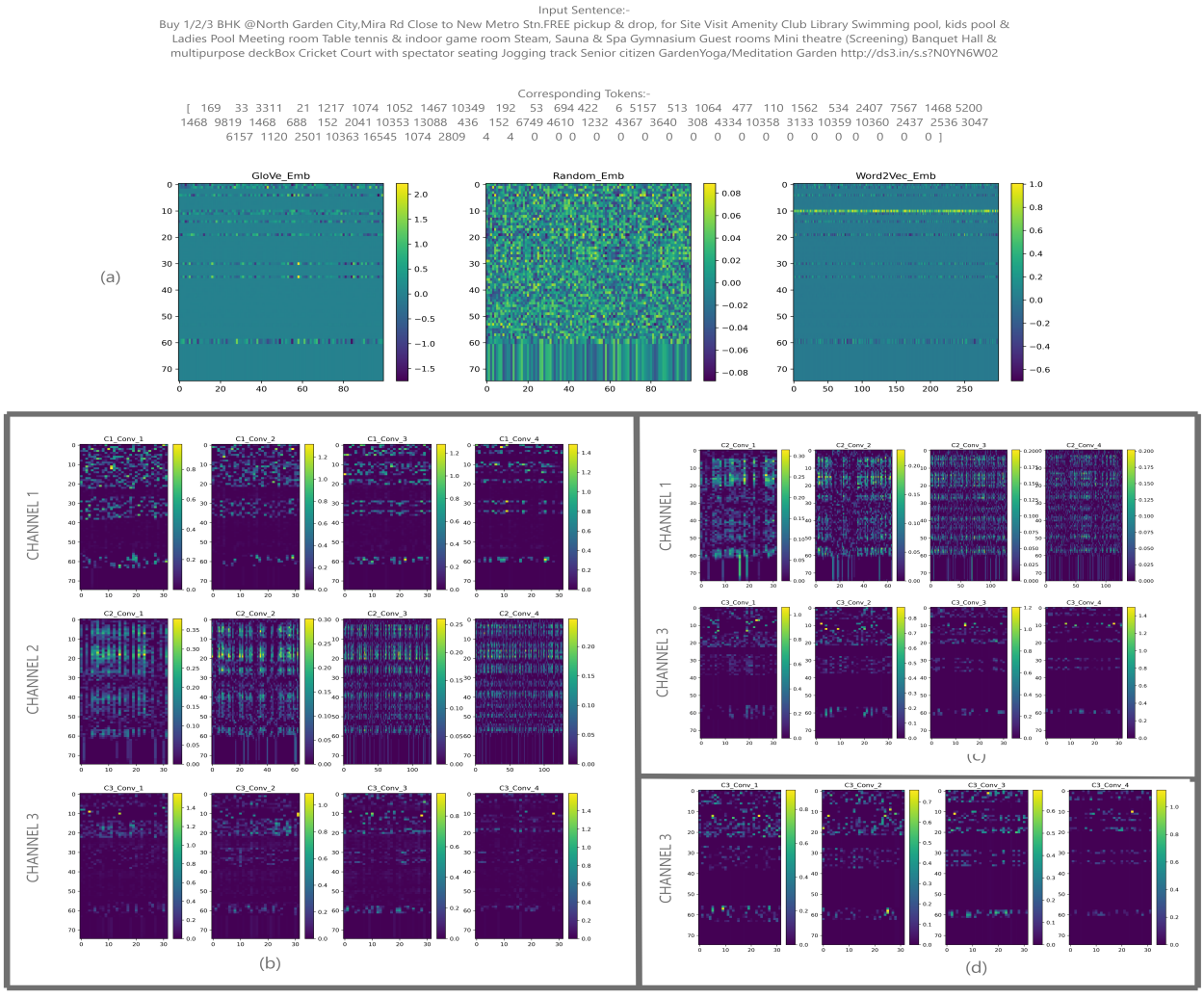


Figure 4: Activation of Convolution layers is arranged in parallel; (a) Word embeddings corresponding to the GloVe, Random, and Word2Vec; (b) Activations of M3P Model (c) Activations of M2P Model (d) Activations of M1P Model

than that required for SMS messages. Secondly, these pre-trained embeddings were kept static to avoid overfitting and therefore are not fine-tuned for the task-at-hand. Finally, as evident in fig.4 (a), in the case of pre-trained embeddings, most of the vectors are initialized to zero and these vectors correspond to those words which are not present in the vocabulary. To put things into perspective, for our dataset out of the 21885 unique tokens present in the data set, Word2vec embeddings were available only for 10797 tokens, and the rest 11088 tokens were initialized to null vectors. Thus we feel that the combination of the aforementioned factors is responsible for the poor performance of the pre-trained embeddings in a stand-alone manner. However, when the same pre-trained embeddings are used in combination with random embeddings, the random embeddings provide some representation for missing words that are non-static and are fine-tuned for the specific task to be performed leading to improved performance.

The next observation which can be made is that as the number of channels increases from 1 to 3, irrespective of the orientation of the convolution layers the accuracy improves. This, according to us, is the effect of obtaining multiple representations of single words which reinforces the n-gram detection capabilities of the convolution filters at the various layers. This can be proved by observing the Activation of the CNN layer in fig. 4 and 5. Fig.4 (c) shows the activations of convolution layers placed in parallel for the M2P model with channel 1 and channel 3 and fig.4 (d) shows the activations of parallel convolution layers for the M1P model with channel 3. From these figures, it can be seen that when channel 3 is used alone, its convolution layers can detect some n-gram patterns with some particular activation levels which are less than or equal to the level of activation for the same layers when used in combination with channel 1 in fig.4 (c) and channel 1 and 2 in fig.4 (b). A similar observation can be made in fig.5 This indicates that multiple channels or embeddings reinforce the activations which help in making more confident predictions. Now since the M3P and M3S models both have random embedding and multiple

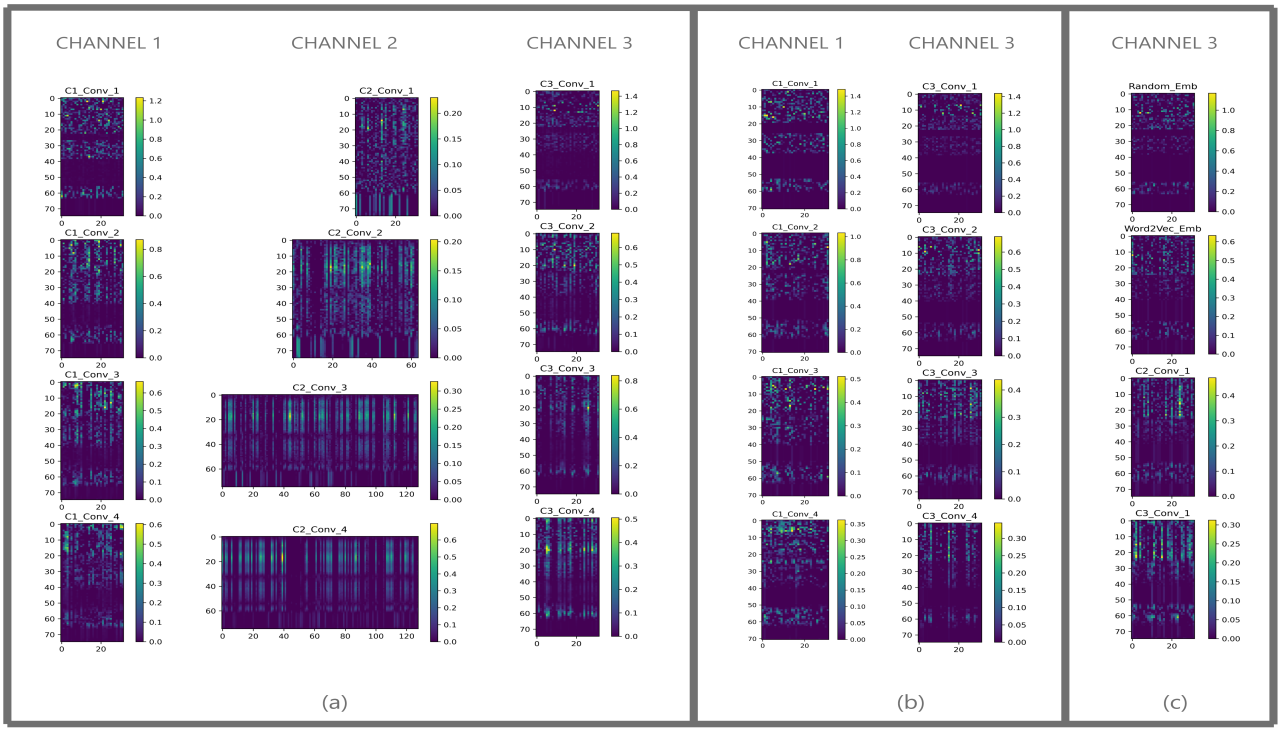


Figure 5: Activation of Convolution layers is arranged in Series; (a) Activations of M3S Model (b) Activations of M2S Model (c) Activations of M1S Model

channels their convolution layers can detect text patterns more confidently leading to better accuracy.

Another important thing that can be observed is as hypothesized earlier, the performance of models with convolution layers in series and convolution in parallel is similar with only slight variation in accuracy occurring due to n-gram level and character level features extracted. Finally, we can see that in the case of single-channel models the difference between the training and testing accuracy is relatively large suggesting over-fitting of the model due to the usage of a single representation. But as the number of channels increases, it is seen that this difference becomes smaller and smaller leading to better testing accuracy and generalization.

Thus, the M3P model which uses 3 channels with static pre-trained and non-static random embeddings, provides enhanced representation of the input, helps learning better and reinforced text pattern detecting filters, and provides better generalization to reduce overfitting and therefore helps obtain the best accuracy out of the variant models.

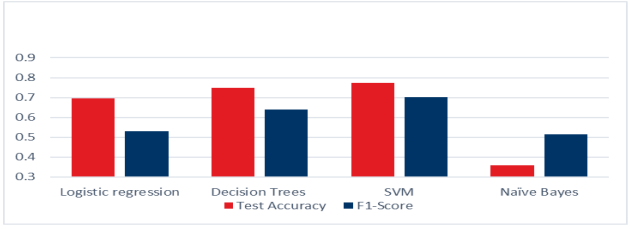


Figure 6: Accuracies and F1 Score of ML Models

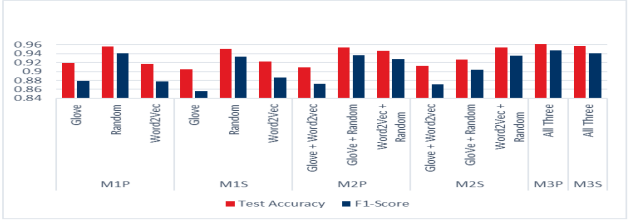


Figure 7: Accuracies and F1 Score of Variant Models

6. Conclusion

In this paper, we have proposed a Multi-Channel CNN architecture with static and non-static embeddings for spam classification. This model was trained on a dataset containing self-collected messages and messages from UCI's spam dataset and we compared it against different machine learning models and variants of the proposed model. This experimentation and comparison provided the following interesting observations. First, using the proposed M3P model and its other variants not only gave better performance than the machine learning algorithms in terms of accuracy, recall, and F1 score but also alleviated the problem of high bias and overfitting. Second, the overall increased performance of the variant models and reduction in overfitting was concomitant with the increase in the number of channels. Visualizing the activations of the convolution layer of different models indicated that the activation strength and hence the confidence

in classification increased with an increase in the number of channels. Third, for the current task, model variants that used non-static random embeddings either alone or in combination with another static pre-trained embedding provided better representation and performance than using only static pre-trained embeddings. Fourth, the orientation of the layers in parallel or series gave more or less similar results indicating performance was independent of orientation in this particular case. Finally, to conclude, the high performance of our proposed model M3P can be attributed to the fact that it was composed of both multiple channels and random embeddings which provided a varied and better representation of the input leading to stronger reinforced activations and better results.

References

Fattahi, J., Mejri, M., 2021. Spaml: a bimodal ensemble learning spam detector based on nlp techniques, in: 2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP), IEEE. pp. 107–112.

Goldberg, Y., 2017. Pre-Trained Word Representation. Morgan amp; Claypool. p. 128.

Jacovi, A., Shalom, O.S., Goldberg, Y., 2020. Understanding convolutional neural networks for text classification. arXiv:1809.08037.

Jain, A.K., Gupta, B.B., 2019. Feature based approach for detection of smishing messages in the mobile environment. Journal of Information Technology Research (JITR) 12, 17–35.

Jain, G., Sharma, M., Agarwal, B., 2019. Optimizing semantic lstm for spam detection. International Journal of Information Technology 11, 239–250.

Kaddoura, S., Chandrasekaran, G., Popescu, D.E., Duraisamy, J.H., 2022. A systematic literature review on spam content detection and classification. PeerJ Computer Science 8, e830.

Kim, Y., 2014. Convolutional neural networks for sentence classification. arXiv:1408.5882.

Luo, Q., Liu, B., Yan, J., He, Z., 2011. Design and implement a rule-based spam filtering system using neural network, in: 2011 International Conference on Computational and Information Sciences, pp. 398–401. doi:10.1109/ICCIS.2011.125.

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., Gao, J., 2021. Deep learning based text classification: A comprehensive review. arXiv:2004.03705.

Mishra, S., Soni, D., 2020. Smishing detector: A security model to detect smishing through sms content analysis and url behavior analysis. Future Generation Computer Systems 108, 803–815.

Parwez, M.A., Abulaish, M., Jahiruddin, 2019. Multi-label classification of microblogging texts using convolution neural network. IEEE Access 7, 68678–68691. doi:10.1109/ACCESS.2019.2919494.

Popovac, M., Karanovic, M., Sladojevic, S., Arsenovic, M., Anderla, A., 2018. Convolutional neural network based sms spam detection, in: 2018 26th Telecommunications Forum (TELFOR), IEEE. pp. 1–4.

Roy, P.K., Singh, J.P., Banerjee, S., 2020. Deep learning to filter sms spam. Future Generation Computer Systems 102, 524–533.

Saidani, N., Adi, K., Allili, M.S., 2020. A semantic-based classification approach for an enhanced spam detection. Computers & Security 94, 101716.

Shrivastava, J.N., Bindu, M.H., 2014. E-mail spam filtering using adaptive genetic algorithm. International Journal of Intelligent Systems and Applications 6, 54–60.

Sonowal, G., Kuppusamy, K., 2018. Smidca: an anti-smishing model with machine learning approach. The Computer Journal 61, 1143–1157.

Yaseen, Q., et al., 2021. Spam email detection using deep learning techniques. Procedia Computer Science 184, 853–858.

Zhang, X., Zhao, J., LeCun, Y., 2015. Character-level convolutional networks for text classification. Advances in neural information processing systems 28.

Zhuang, X., Zhu, Y., Peng, Q., Khurshid, F., 2021. Using deep belief network to demote web spam. Future Generation Computer Systems 118, 94–106.