

**SPEAR- PHISHING ATTACK DETECTION USING  
ARTIFICIAL INTELLIGENCE**

By

**RAJKUMARADEVAN A/L SANGLIDEVAN**

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

**BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMMUNICATIONS  
AND NETWORKING**

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2024

## REPORT STATUS DECLARATION FORM

**Title:** SPEAR- PHISHING ATTACK DETECTION USING ARTIFICIAL INTELLIGENCE

**Academic Session:** 202406

I RAJKUMARDEVAN A/L SANGLIDEVAN

**(CAPITAL LETTER)**

declare that I allow this Final Year Project Report to be kept in

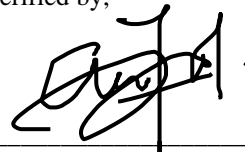
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.



(Author's signature)

Verified by,



(Supervisor's signature)

**Address:**

NO.6, JALAN SATU, TAMAN

KENARI.35000 TAPAH

PERAK

PUAN NOR 'AFIFAH BINTI SABRI

Supervisor's name

**Date:** 12/09/2024

**Date:** 12/09/2024

<b>Universiti Tunku Abdul Rahman</b>			
Form Title : <b>Sample of Submission Sheet for FYP/Dissertation/Thesis</b>			
Form Number: <b>FM-IAD-004</b>	Rev No.: <b>0</b>	Effective Date: <b>21 JUNE 2011</b>	Page No.: <b>1 of 1</b>

**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY  
UNIVERSITI TUNKU ABDUL RAHMAN**

Date: 12/09/2024

**SUBMISSION OF FINAL YEAR PROJECT**

It is hereby certified that RAJKUMARADEVAN A/L SANGLIDEVAN (ID No: 16ACB03600) has completed this final year project entitled “SPEAR- PHISHING ATTACK DETECTION USING ARTIFICIAL INTELLIGENCE” under the supervision of PUAN NOR 'AFIFAH BINTI SABRI (Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.


Yours truly,



\_\_\_\_\_  
(RAJKUMARADEVAN A/L SANGLIDEVAN)

## DECLARATION OF ORIGINALITY

I declare that this report entitled “**SPEAR- PHISHING ATTACK DETECTION USING ARTIFICIAL INTELLIGENCE**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :  \_\_\_\_\_

Name : RAJKUMARDEVAN A/L SANGLIDEVAN

Date : 12/9/2024

## **ACKNOWLEDGEMENTS**

First and foremost, I express my heartfelt appreciation to my project supervisor, Puan Nor 'Afifah Binti Sabri, for his unwavering dedication, insightful feedback, valuable insights, practical recommendations, and continuous inspiration, all of which have significantly contributed to my project and the completion of this paper.

However, none of this would have been possible without the generous support and assistance of numerous individuals and organizations.

I am deeply grateful to my family for their unwavering support and comforting presence throughout this endeavour. Additionally, I extend my sincere thanks to professionals in the business community for their patience and motivational words.

I also extend my appreciation and admiration to my model advisor and others who have generously shared their expertise and assistance in various capacities

## **ABSTRACT**

This project focuses on developing machine learning-based applications to enhance cybersecurity, specifically the Spear Phishing Attack Detection (S.P.A.D) system and the Email/SMS Classifier. The goal is to mitigate phishing and spam threats by using advanced algorithms to detect malicious URLs and classify messages effectively. The Spear Phishing Attack Detection system employs models such as Logistic Regression, Random Forest, and ensemble methods to identify and block phishing URLs. It provides real-time feedback on website safety, offering a proactive defense against spear phishing attacks. Extensive testing confirmed the system's accuracy in correctly classifying phishing and legitimate URLs. The Email/SMS Classifier uses models like Naive Bayes, Support Vector Machines, and Random Forest to classify messages as spam or legitimate. The system integrates text preprocessing techniques to enhance classification accuracy and was tested with real-world datasets, demonstrating effective spam detection. Both applications underwent thorough functional, performance, and accuracy testing. Metrics such as precision, recall, and F1 score were used to evaluate effectiveness. The systems were also tested for performance and scalability to handle large data volumes without sacrificing speed or accuracy. The project also explores the characteristics of spear phishing and spam, offering insights into attackers' evolving tactics. These findings inform the development of stronger cybersecurity defenses. Recommendations for future work include refining the models, expanding datasets, and continuously updating systems to adapt to new threats. By integrating these applications into broader security frameworks, their impact could be further enhanced. In summary, this project successfully demonstrates how machine learning can be used to detect and prevent spear phishing and spam, offering innovative solutions to enhance cybersecurity for individuals and organizations.

# TABLE OF CONTENTS

<b>TITLE PAGE</b>	<b>i</b>
<b>REPORT STATUS DECLARATION FORM</b>	<b>ii</b>
<b>FYP THESIS SUBMISSION FORM</b>	<b>iii</b>
<b>DECLARATION OF ORIGINALITY</b>	<b>iv</b>
<b>ACKNOWLEDGEMENTS</b>	<b>v</b>
<b>ABSTRACT</b>	<b>vi</b>
<b>TABLE OF CONTENTS</b>	<b>vii</b>
<b>LIST OF FIGURES</b>	<b>x</b>
<b>LIST OF TABLES</b>	<b>xi</b>
<b>LIST OF SYMBOLS</b>	<b>xii</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xiii</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Problem Statement and Motivation	1
1.2 Project Objectives	2
1.3 Project Scope	3
1.4 Contributions	4
1.5 Report Organization	5
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>6</b>
2.1 Review of the Technologies	6
2.1.1 Hardware Platform	6
2.1.2 Firmware/OS	7
2.1.3 Database	8
2.1.4 Programming Language	9
2.1.5 Algorithm	10
2.1.6 Summary of the Technologies Review	12
2.2 Existing Technique on Machine Learning	13
2.2.1 Strengths and Weakness	15
2.2.2 Summary of Existing Technique	18
2.3 Concluding Remark	20

<b>CHAPTER 3 SYSTEM METHODOLOGY/APPROACH</b>	<b>21</b>
3.1 System Development Models	21
3.1.1 Waterfall Model	24
3.1.2 Agile Model	25
3.1.3 Spiral Model	26
3.1.4 V-Model (Verification and Validation Model)	27
3.2 Proposed Method/Approach	27
3.3 System Requirement	29
3.3.1 Hardware	29
3.3.2 Software	30
3.4 System Architecture/Design Diagram	33
3.5 Expected Challenges	39
3.6 Project Milestone	40
3.7 Concluding Remarks	41
<b>CHAPTER 4 SYSTEM DESIGN</b>	<b>42</b>
4.1 System Architecture	42
4.2 Functional Modules in the System	44
4.3 System Flowchart	47
4.3.1 System Flowchart (S.P.A.D Plugin)	49
4.3.2 System Flowchart (Email\SMS Classifier)	50
4.4 Concluding Remark	51
<b>CHAPTER 5 SYSTEM IMPLEMENTATION</b>	<b>52</b>
5.1 Software Setup	52
5.2 Setting and Configuration	56
5.3 System Operation (with Screenshot)	61
5.4 Implementation Issues and Challenges	65
5.5 Concluding Remark	66
<b>CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION</b>	<b>67</b>
6.1 System Testing and Performance Metrics	67
6.2 Testing Setup and Result	69



6.2.1	Testing Setup for S.P.A.D Plugin Application	69
6.2.2	Testing Setup for Email/SMS Classifier Application	71
6.3	Project Challenges	74
6.4	Objectives Evaluation	75
6.5	Concluding Remark	77
<b>CHAPTER 7 CONCLUSION AND RECOMMENDATION</b>		<b>79</b>
7.1	Conclusion	79
7.2	Recommendation	80
<b>REFERENCES</b>		<b>84</b>
<b>APPENDIX</b>		<b>85</b>
<b>WEEKLY LOG</b>		<b>85</b>
<b>POSTER</b>		<b>89</b>
<b>PLAGIARISM CHECK RESULT</b>		<b>90</b>
<b>FYP2 CHECKLIST</b>		<b>93</b>

## LIST OF FIGURES

<b>Figure Number</b>	<b>Title</b>	<b>Page</b>
Figure 3-1	System Architecture Diagram (S.P.A.D Plugin)	33
Figure 3-2	System Architecture Diagram (Email/SMS Classifier)	36
Figure 3-3	The Gantt Chart for FYP1	40
Figure 3-4	The Gantt Chart for FYP2	40
Figure 4-1	Block Diagram (S.P.A.D Plugin)	42
Figure 4-2	Block Diagram (Email/SMS Classifier)	43
Figure 4-3	System Flowchart (S.P.A.D Plugin)	47
Figure 4-4	System Flowchart (Email\SMS Classifier)	48
Figure 5-1	S.P.A.D Plugin Shows Loading Message	62
Figure 5-2	S.P.A.D Plugin Shows the Website Is Safe	63
Figure 5-3	S.P.A.D Plugin Shows the Website Is Suspicious	63
Figure 5-4	S.P.A.D Plugin Shows the Website Is Phishing	64
Figure 5-5	Email/SMS Classifier Interface	64
Figure 6-1	S.P.A.D Plugin Shows the Website Is Safe to Use	70
Figure 6-2	S.P.A.D Plugin Shows the Suspicious Website May Not Be Safe to Use	70
Figure 6-3	S.P.A.D Plugin Shows the Phishing Website May Not Be Safe to Use	71
Figure 6-4	Graph Results for Accuracy and Precision Using Several Classifier	72
Figure 6-5	Summary Results for Accuracy and Precision Using Several Classifier	72
Figure 6-6	Results for Accuracy and Precision Using SVM, NB and ET	73
Figure 6-7	Results For Email/SMS Classifier Showing Not Spam	73
Figure 6-8	Results For Email/SMS Classifier Showing Spam	73

## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 2-1	Summary of Existing Technique	18
Table 3-1	Specifications of laptop	29

## LIST OF SYMBOLS

Symbol	Explanation
%	Percentage
'	Apostrophes
-	Hyphen
“”	Double Quotation Marks

## LIST OF ABBREVIATIONS

<b>Abbreviations</b>	<b>Explanation</b>
<i>IP</i>	Internet Protocol
<i>URL</i>	Uniform Resource Locators
<i>TLD</i>	Top-Level Domains
<i>APT</i>	Advanced Persistent Threat
<i>APWG</i>	Anti-Phishing Working Group
<i>RSA</i>	Rivest, Shamir, Adleman
<i>SMS</i>	Short Message Service
<i>SVM</i>	Support Vector Machine
<i>NB</i>	Naive Bayes Classifier
<i>ET</i>	Extra Trees Classifier
<i>RF</i>	Random Forest
<i>PKL</i>	Pickle
<i>TWSVM</i>	Twin Support Vector Machine
<i>APT</i>	Advanced Persistent Threats

# CHAPTER 1

## Introduction

### 1.1 Problem Statement and Motivation

In 2024, the Anti-Phishing Working Group (APWG) released its annual report, unveiling a staggering count of over 1,077,501 phishing attacks in the fourth quarter of 2023[1]. Concurrently, an analysis conducted by RSA revealed that organizations worldwide incurred losses totalling \$9 billion in 2023 alone due to phishing attacks. These figures underscore a concerning trend, indicating the inadequacy of existing anti-phishing tools and initiatives in mitigating the pervasive threat posed by cybercriminals. Personal computer users often find themselves susceptible to phishing attacks for several key reasons: a lack of fundamental comprehension regarding Uniform Resource Locators (URLs), uncertainty regarding the trustworthiness of web pages, ambiguity surrounding page locations due to redirection or obscured URLs, the prevalence of deceptive URLs, and inadvertent navigation to fraudulent pages. Compounding these challenges is the difficulty users encounter in distinguishing phishing websites from legitimate ones, further exacerbating their vulnerability to exploitation.

Phishing websites serve as common entry points for online social engineering endeavours, facilitating a myriad of fraudulent activities perpetrated by malicious actors. Attackers adeptly replicate legitimate websites, disseminating suspicious URLs via spam messages, SMS, or social networking platforms to ensnare unsuspecting victims. Leveraging email, phone calls, or text messages, attackers propagate counterfeit versions of authentic websites, luring victims into divulging personal or highly sensitive information such as banking credentials or government-issued identification numbers. These nefarious tactics erode trust in social services, undermining confidence in online platforms and services.

The pernicious consequences of spear phishing extend beyond mere data breaches, encompassing the grave implications of unauthorized access to users' sensitive information, resulting in substantial financial losses and potentially depriving individuals of access to their own accounts. Thus, this study endeavours to undertake a

## CHAPTER 1

comprehensive analysis of spear phishing features, seeking to categorize and evaluate them systematically to effectively thwart and mitigate the inherent dangers posed by spear phishing assaults. Furthermore, the integration of advanced clustering algorithms will be employed to augment the efficacy of detection mechanisms, thereby bolstering the resilience of cybersecurity measures against the increasingly sophisticated tactics employed by malicious actors in spear phishing campaigns.

### 1.2 Project Objectives

Following are the main objectives of this project.

- To propose innovative strategies and methodologies for enhancing the detection and mitigation of spear phishing attacks in cyberspace.
- Leverage advanced machine learning techniques and algorithmic frameworks to develop robust solutions capable of identifying and thwarting spear phishing threats effectively.
- To explore the underlying mechanisms and characteristics of spear phishing assaults to shed light on the intricate dynamics at play.
- To inform the development of proactive countermeasures against spear phishing through empirical analysis and experimentation.
- Contribute valuable insights to the field of cybersecurity to foster a safer and more secure digital ecosystem for individuals and organizations.
- Exclude tangential topics such as general phishing attacks unrelated to spear phishing and unrelated cybersecurity domains like network intrusion detection or malware analysis.
- This project will not specifically address legal or regulatory aspects of cybersecurity or the broader socio-political implications of cyber threats.

### 1.3 Project Scope

The scope of this project encompasses an in-depth investigation into spear phishing attacks and the development of comprehensive countermeasures to mitigate their impact. Key objectives include:

- ❖ **Analysis of Spear Phishing Tactics:** Conducting a thorough examination of the various techniques employed by cybercriminals in spear phishing attacks, including social engineering tactics, email spoofing, and website impersonation.
- ❖ **Identification of Spear Phishing Features:** Identifying and categorizing the distinguishing features of spear phishing emails, websites, and communication channels to facilitate effective detection and classification.
- ❖ **Development of Detection Mechanisms:** Designing and implementing machine learning algorithms and clustering techniques to detect and classify spear phishing attempts with high accuracy and efficiency.
- ❖ **Evaluation of Detection Performance:** Conducting rigorous testing and evaluation of the developed detection mechanisms using real-world datasets and simulated spear phishing scenarios to assess their effectiveness and reliability.
- ❖ **Integration of Countermeasures:** Exploring the integration of proactive countermeasures, such as email filtering systems, browser extensions, and user education initiatives, to augment the overall resilience against spear phishing attacks.
- ❖ **Documentation and Reporting:** Compiling the research discoveries, methodologies employed, and implementation specifics into an extensive report with the aim of offering insights and suggestions for forthcoming research endeavors and practical implementations within the cybersecurity domain.



## 1.4 Contributions

This project aims to make significant contributions to the field of cybersecurity through the following avenues:

- ❖ **Advancement of Detection Techniques:** By developing novel machine learning algorithms and clustering methodologies tailored specifically for spear phishing detection, the project seeks to advance the state-of-the-art in cybersecurity research. These techniques are expected to improve the accuracy and efficiency of detecting spear phishing attacks, thereby enhancing overall cybersecurity resilience.
- ❖ **Empirical Validation and Evaluation:** The project will contribute to empirical research in cybersecurity by rigorously testing and evaluating the developed detection mechanisms using real-world datasets and simulated spear phishing scenarios. The results of these evaluations will provide empirical evidence of the effectiveness and reliability of the proposed techniques.
- ❖ **Practical Applications and Implications:** The findings and methodologies developed in this project have practical implications for cybersecurity practitioners, policymakers, and organizations. Insights gained from the research can be used to inform the design and implementation of cybersecurity solutions, training programs, and policy frameworks aimed at combating spear phishing attacks and enhancing overall cyber resilience.
- ❖ **Knowledge Dissemination and Awareness:** Through the documentation and dissemination of research findings, the project aims to raise awareness about the evolving threat landscape of spear phishing and the importance of adopting proactive cybersecurity measures. By sharing knowledge and insights with the broader cybersecurity community, the project seeks to contribute to collective efforts in safeguarding digital assets and protecting against cyber threats.

### **1.5 Report Organization**

The report is structured to explore the development of a spear phishing attack detection system using artificial intelligence, with each chapter addressing critical aspects of the research. Chapter 2 reviews existing literature on AI-based spear phishing detection. It examines current detection methodologies, AI algorithms, and the latest advancements in combating phishing threats. This chapter highlights emerging trends and promising areas for future investigation. Chapter 3 outlines the methodology used in developing the spear phishing detection system. This includes data collection, feature engineering, model training, and evaluation. The chapter offers a detailed explanation of the process, ensuring the system's robustness and accuracy. Chapter 4 delves into the design and architecture of the detection system, focusing on integrating AI algorithms for real-time detection. It discusses key technical components like feature extraction, model architecture, and the deployment strategies used to ensure efficient system performance. Chapter 5 discusses the system implementation, detailing the integration of machine learning models, backend server development, and the browser plugin's functionality. It explains how users can scan URLs and emails for phishing attempts and outlines key challenges such as model accuracy, performance, and security during implementation. Chapter 6 covers the testing and evaluation of the system. Functional, performance, and accuracy tests are conducted, and metrics like precision, recall, and F1 scores are used to assess the system's performance. These tests ensure the system operates reliably under various conditions. Chapter 7 concludes the report, summarizing the project's outcomes and offering recommendations for future improvements. Suggestions include refining the models, expanding datasets, and incorporating user feedback to enhance the system's detection capabilities. The chapter also highlights potential future applications and ongoing updates to keep pace with evolving phishing techniques. The report presents a comprehensive overview of the research, design, implementation, and evaluation of a spear phishing detection system and its real-world cybersecurity implications.

## CHAPTER 2

### Literature Review

#### **2.1 Review of the Technologies**

The development of the Spear Phishing Attack Detection system hinges on the integration of various technologies that ensure the system's overall effectiveness, scalability, and reliability. These technologies form the backbone of the system, enabling it to function seamlessly across multiple environments while delivering accurate and efficient phishing detection. In this section, we will examine the core components, including the hardware platform, operating system, database, programming language, and algorithms used in the design and implementation of the system.

##### **2.1.1 Hardware Platform**

The hardware platform is a crucial consideration in the design and deployment of phishing detection systems. In many cases, standard computing systems are sufficient for the initial stages of model development, such as training machine learning algorithms. However, as the complexity of these models increases and the volume of data to be processed grows, the need for more powerful hardware becomes apparent.

For example, real-time phishing detection systems like PhishStorm require high-performance servers to handle the large-scale implementation of real-time URL analysis. These systems must process vast amounts of data quickly and accurately, necessitating the use of servers equipped with multi-core processors, large amounts of RAM, and fast storage solutions. In addition, the deployment of these systems in cloud environments can further enhance their scalability and performance. Cloud computing platforms like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud offer scalable infrastructure that can dynamically allocate resources based on the system's requirements. This flexibility allows for the efficient handling of fluctuating workloads, particularly during peak times when phishing attacks may surge.

Moreover, the choice of hardware also impacts the energy efficiency and operational costs of phishing detection systems. High-performance servers can be energy-intensive,

and organizations need to consider the trade-off between performance and cost. Emerging technologies like edge computing can help mitigate these challenges by processing data closer to the source, reducing latency and energy consumption. This approach can be particularly beneficial in scenarios where real-time decision-making is critical, such as detecting phishing attempts in emails or web traffic.

In summary, while standard computing systems may suffice for initial development, the deployment of large-scale, real-time phishing detection systems often requires high-performance hardware, cloud infrastructure, and innovative approaches like edge computing to meet the demands of modern cybersecurity.

### **2.1.2 Firmware/OS**

The firmware and operating system (OS) environments in which phishing detection systems are deployed can significantly impact their performance, compatibility, and security. Although specific firmware/OS details are generally not highlighted in the literature, it is essential to recognize that the underlying system environment plays a critical role in the execution of machine learning models and the overall functioning of the detection system.

Phishing detection models can be implemented across various operating systems, including Linux, Windows, and macOS, depending on the development environment and the target deployment infrastructure. Linux is often favored in research and production environments due to its open-source nature, flexibility, and stability. It is particularly well-suited for running machine learning models on servers, where performance and security are paramount. Additionally, Linux's compatibility with various programming languages, libraries, and tools used in phishing detection research, such as Python, TensorFlow, and scikit-learn, makes it a popular choice among developers.

On the other hand, Windows may be preferred in environments where integration with other Windows-based applications and services is required. For instance, organizations that rely heavily on Microsoft Office 365 or Exchange Server may opt for phishing detection systems that are compatible with the Windows ecosystem. Windows also

offers a user-friendly interface and robust support for enterprise security features, making it a viable option for organizations with specific requirements.

Moreover, the choice of firmware/OS also influences the security of the phishing detection system. Firmware vulnerabilities can be exploited by attackers to gain unauthorized access to the system, potentially compromising the detection process. Therefore, it is crucial to keep both firmware and OS up to date with the latest security patches and updates.

In conclusion, while the specific firmware/OS environments may not always be explicitly mentioned in phishing detection research, their impact on system performance, compatibility, and security should not be overlooked. The choice of OS should align with the system's requirements, the organization's infrastructure, and the need for robust security measures.

### **2.1.3 Database**

The database is a vital component of phishing detection systems, serving as the repository for storing and managing the data used in training, testing, and deploying machine learning models. The databases used in phishing detection research vary significantly, from URL datasets to spam email databases, each tailored to the specific requirements of the detection approach.

Efficient database management is crucial for real-time systems like PhishStorm, which need to handle large volumes of URL and query data dynamically. These systems must be capable of quickly retrieving and analyzing data to provide timely detection of phishing attempts. Relational databases like MySQL and PostgreSQL are commonly used for structured data, offering powerful querying capabilities and support for complex data relationships. These databases are well-suited for storing URL features, user behavior data, and historical phishing incidents, enabling the system to learn from past attacks and improve its detection accuracy over time.

However, as the volume of data grows, traditional relational databases may struggle to keep up with the demands of real-time processing. In such cases, NoSQL databases like MongoDB and Cassandra offer a more scalable solution, allowing for the storage and retrieval of unstructured or semi-structured data at high speeds. These databases are particularly useful in handling large-scale datasets, such as those generated by web

traffic or social media, where phishing attempts may be detected through patterns in the data.

Furthermore, the integration of real-time data streams with the database is essential for maintaining the system's effectiveness. Tools like Apache Kafka and Amazon Kinesis can be used to stream data in real time, ensuring that the database is continuously updated with the latest information. This capability is particularly important in dynamic environments where phishing tactics evolve rapidly, and the system must adapt to new threats as they emerge.

In addition to managing data storage and retrieval, the database also plays a critical role in data preprocessing and feature extraction. Before feeding data into machine learning models, it must be cleaned, normalized, and transformed into a format suitable for analysis. This preprocessing step is essential for ensuring the quality and consistency of the data, which directly impacts the accuracy of the detection system.

In summary, the choice of database and the efficiency of its management are critical to the success of phishing detection systems. Whether using relational databases for structured data or NoSQL databases for large-scale, unstructured data, the system must be capable of handling the demands of real-time processing, data streaming, and preprocessing to deliver accurate and timely detection results.

### **2.1.4 Programming Language**

The choice of programming language is a crucial factor in the development of phishing detection systems, as it influences the ease of implementation, performance, and compatibility with various tools and libraries. In the field of machine learning and phishing detection, Python and R are among the most used programming languages due to their extensive libraries, community support, and ease of use.

Python is particularly popular in phishing detection research, as it offers a wide range of libraries and frameworks for machine learning, data analysis, and web development. Libraries such as TensorFlow, Keras, and scikit-learn provide powerful tools for building and training machine learning models, while Pandas and NumPy are essential for data manipulation and preprocessing. Python's versatility also extends to web scraping and natural language processing (NLP), which are often required in phishing

## CHAPTER 2

detection tasks. For example, BeautifulSoup and Selenium can be used to scrape web pages for phishing indicators, while NLTK and spaCy enable the analysis of email content to detect phishing attempts.

R is another popular language in the data science community, particularly for statistical analysis and visualization. It offers robust support for machine learning through packages like caret and random Forest, making it a valuable tool for researchers focused on developing and testing new algorithms. R's strong visualization capabilities also make it useful for analyzing and presenting the results of phishing detection experiments.

In addition to Python and R, other programming languages may be used depending on the specific requirements of the project. JavaScript is mentioned in some approaches for phishing attacks, particularly in the context of web-based detection systems. JavaScript can be used to develop browser extensions or client-side scripts that analyze web pages in real time, detecting phishing attempts as users browse the internet.

Java is another language that may be employed in phishing detection systems, especially in environments requiring robust, enterprise-level solutions. Java's scalability, performance, and security features make it a suitable choice for large-scale deployments, such as server-based detection systems or integrations with existing enterprise security platforms.

Overall, the choice of programming language in phishing detection systems is guided by the specific needs of the project, the available libraries and tools, and the developer's expertise. Python and R remain the dominant languages in the field due to their extensive support for machine learning and data analysis, but other languages like JavaScript and Java may also play important roles in specific contexts.

### **2.1.5 Algorithm**

The algorithms used in phishing detection systems are at the core of their functionality, enabling the identification of phishing attempts through various techniques. Several machine learning and heuristic algorithms are commonly utilized across different studies, each offering unique strengths and trade-offs.

1. **Naive Bayes (NB):** Naive Bayes is a simple, yet effective probabilistic classifier based on Bayes' theorem. It assumes that features are independent of each other, which simplifies the computation. Despite its simplicity, Naive Bayes has shown good performance in phishing detection tasks, particularly in text-based analysis where the presence of certain words or phrases can indicate phishing attempts.
2. **Support Vector Machines (SVM):** SVM is a powerful supervised learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that separates data points into different classes. In phishing detection, SVM has been widely used due to its ability to handle high-dimensional data and its effectiveness in binary classification problems. However, SVM can be computationally expensive, especially with large datasets, which may limit its use in real-time systems.
3. **Random Forest (RF):** Random Forest is an ensemble learning method that combines multiple decision trees to improve classification accuracy. It is particularly useful in phishing detection due to its robustness to overfitting and its ability to handle many features. RF can automatically rank the importance of features, making it easier to identify the most relevant indicators of phishing. However, the model's complexity can make it challenging to interpret, which may be a drawback in some applications.
4. **Twin Support Vector Machines (TWSVM):** TWSVM is a variant of SVM that constructs two non-parallel hyperplanes to classify data points. This approach can improve classification accuracy by considering both the positive and negative classes separately. In phishing detection, TWSVM has shown promise in handling imbalanced datasets, where phishing attempts are relatively rare compared to legitimate activities. However, like SVM, TWSVM can be computationally intensive and may require significant resources for training and deployment.
5. **Bayesian Algorithms:** Bayesian algorithms are probabilistic models that use Bayes' theorem to update the probability of a hypothesis based on new evidence. These algorithms are particularly useful in phishing detection because they can incorporate prior knowledge and adapt to new data over time. Bayesian



networks, in particular, have been used to model the relationships between different features, providing a more nuanced approach to classification. However, Bayesian algorithms can be sensitive to the quality of the input data, and their performance may degrade if the data is noisy or incomplete.

6. **Heuristic Approaches:** Heuristic approaches rely on predefined rules or patterns to detect phishing attempts. These methods are often used in combination with machine learning algorithms to improve detection accuracy. For example, a heuristic approach may involve checking for the presence of suspicious URLs, unusual email addresses, or misleading visual elements on a website. While heuristics can be effective in detecting known phishing techniques, they may struggle to adapt to new or sophisticated attacks. Additionally, heuristic methods can generate false positives, as legitimate websites or emails may inadvertently trigger the predefined rules.

In conclusion, the choice of algorithm in phishing detection systems depends on various factors, including the nature of the data, the required accuracy, and the available computational resources. Machine learning algorithms like SVM, Random Forest, and TWSVM offer powerful tools for detecting phishing attempts, but they may require significant resources and careful tuning to achieve optimal performance. On the other hand, heuristic approaches provide a more straightforward solution but may lack the flexibility needed to detect evolving threats. Combining multiple algorithms and techniques, such as using heuristics alongside machine learning models, can help create a more robust and adaptable phishing detection system.

### **2.1.6 Summary of the Technologies Review**

The Spear Phishing Attack Detection system relies on a diverse array of technologies to achieve its goals of efficiency, scalability, and accuracy. The hardware platform includes high-performance servers and cloud computing solutions, such as AWS and Azure, which are essential for managing large datasets and real-time processing needs. Edge computing technologies further optimize performance by reducing latency and energy consumption. The choice of firmware and operating systems also impacts system performance and security. Linux is favored for its stability and compatibility with machine learning tools, while Windows is selected for environments needing

integration with Microsoft services. Keeping firmware and OS updated is crucial for maintaining security.

Databases play a key role in managing the data used for training and deploying the detection models. Relational databases like MySQL and PostgreSQL handle structured data, whereas NoSQL databases such as MongoDB and Cassandra manage large-scale, unstructured data efficiently. Real-time data streaming tools like Apache Kafka and Amazon Kinesis are employed to ensure that the system adapts quickly to emerging phishing threats.

Programming languages like Python and R are extensively used due to their powerful libraries for machine learning and data analysis, with Python also supporting web scraping and NLP tasks. JavaScript and Java are utilized for specific applications, such as web-based or enterprise-level solutions.

The choice of algorithms is critical for effective phishing detection. Techniques including Naive Bayes, Support Vector Machines, Random Forest, Twin Support Vector Machines, Bayesian methods, and heuristic approaches each bring unique strengths to the table. Together, these technologies ensure the system can accurately identify and adapt to evolving phishing threats, maintaining high performance and reliability.

### **2.2 Existing Technique on Machine Learning**

1. Xiaoqing et al. (2018) demonstrated an intelligent automated approach for detecting phishing webpages[4]. They looked into the characteristics of a uniform resource locator (URL) and classified them using NB. In the instance of dubious webpages, SVM is used to parse and reclassify them. They claim that the technology provides high detection accuracy in less time based on their findings. A similar strategy was utilized by Moghimi, M., and Varjani, A.Y. (2016), who described a method that combined SVM and decision tree models[11]. SVM is utilized for training, and a decision tree is used to create rules for identifying phishing websites that target the banking domain.

2. Almseidin et al. (2019) published a research in which they improved the performance of their system by using several machine learning algorithms and feature selection approaches[3]. The tests were carried out using a 48-feature phishing dataset that included 5000 benign and phishing webpages. They came to the conclusion that the RF method, which has only 20 characteristics, provides the highest accuracy.
3. Rao R. S. et al (2015) developed a heuristic technique employing TWSVM (twin support vector machine) classifier to detect harmful enrolled phishing websites and moreover websites that are facilitated on arrangement servers[13]. Their technique looks at the sign-in page and the main page of the visiting site to differentiate phishing sites housed on legitimate domains. The hyperlink and URL-based characteristics are used to identify phishing websites that are maliciously enlisted. They used a variety of support vector machines to set up phishing websites (SVMs). They discovered that the TWSVM (twin support vector machine classifier) outperforms other modifications.
4. Jain & Gupta, (2017), proposed an online search tool-based approach for detecting phishing site pages with pinpoint accuracy, regardless of the literary language used on the page[8]. To determine the legitimacy of the suspicious URL, the suggested online search tool-based approach employs a lightweight, trustworthy, and language-independent pursuing enquiry. They've also combined five heuristics with the web search tool-based instrument to improve recognition accuracy. This is because some newly created legitimate sites may not appear in the web index. The suggested technique may also be used to organize newly created lawful websites that are not yet classified by readily available internet searcher-based tactics.
5. Marchal et al. (2014) proposed a system PhishStorm which is an automated phishing detection system that can examine any URL in real time to detect probable phishing sites[5]. To protect users from phishing material, Phish storm is presented as an automatic real-time URL phishing grading system. PhishStorm used as a Website reputation evaluation system that delivers a phishing rating for URLs. PhishStorm scrutinizes the structure of URLs, distinguishing between the registered low-level domain and the remaining upper-level domain, path, and query

components. Leveraging the concept of intra-URL relatedness, it assesses features derived from URL words, utilizing query data sourced from popular search engines like Google or Yahoo.

6. Baykara & Gurel, (2018) proposed the Anti-Phishing Simulator, an application that provides information on the phishing detection difficulty and how to identify phishing emails[9]. The Bayesian algorithm adds spam emails to the database. Phishing attackers utilise JavaScript to insert a valid URL into the address bar of the browser. The study suggests that you only utilise the content of the e-mail as a keyword when performing complicated word processing.
7. Che H. et al. (2017) relied on the elements that the text of these communications refers to in order to identify phishing messages[6]. These objects are referred to as circumstances, and a pair of circumstances is referred to as an occurrence pair, which addresses the relationship between the two circumstances, and the semantic web is used to convert words in messages to circumstances. Their research will provide another computation for differentiating phishing communications that rely on occurrence matching. The first element of this computation is to create a semantic web knowledge base, which provides relationships between words and occurrences. The next step is to create the categorization information base, which will be used to organise phishing communications. The final section explains how to use the semantic web knowledge base and class data set to detect phishing.

### **2.2.1 Strengths and Weakness**

#### **Xiaoqing et al. (2018)**

**Strengths:** Utilization of both Naive Bayes (NB) and Support Vector Machine (SVM) algorithms for classification, enhancing detection accuracy. Integration of intelligent automated approach for detecting phishing webpages, demonstrating high accuracy and efficiency.

**Weaknesses:** Machine learning models like Naive Bayes and Support Vector Machines rely on training data to generalize patterns and make predictions. If the training data is

## CHAPTER 2

not diverse or representative enough of real-world phishing scenarios, the model's performance may suffer, leading to false positives or false negatives.

### **Almseidin et al. (2019)**

Strengths: Improvement of system performance through the utilization of multiple machine learning algorithms and feature selection approaches.

Achievement of high accuracy using Random Forest (RF) method with only 20 characteristics, indicating efficient feature selection.

Weaknesses: While the dataset used in the research consists of 5000 benign and phishing webpages, the size may still be relatively small considering the complexity and diversity of real-world phishing attacks. Additionally, the composition of the dataset (e.g., distribution of phishing vs. benign URLs, types of phishing attacks represented) could impact the generalizability of the findings.

### **Rao R. S. et al. (2015)**

Strengths: Development of a heuristic technique employing Twin Support Vector Machine (TWSVM) classifier for detecting harmful phishing websites, demonstrating high accuracy. Consideration of both sign-in page and main page characteristics for differentiation of phishing sites, enhancing detection capability.

Weaknesses: Heuristic techniques rely on rules of thumb or educated guesses to solve problems, rather than strictly following algorithms. While heuristics can be effective in certain contexts, they may not always provide optimal solutions and could be prone to overlooking important patterns or features in the data

### **Jain & Gupta, (2017),**

Strengths: Proposal of an online search tool-based approach for detecting phishing site pages with pinpoint accuracy, irrespective of language barriers. Integration of heuristics with web search tool-based instrument to enhance detection accuracy, indicating a comprehensive detection strategy.

Weaknesses: Potential reliance on external search engines for determining the legitimacy of URLs, which may introduce dependency on external factors and sources.

## CHAPTER 2

Combining multiple heuristics with the web search tool-based approach can potentially enhance detection accuracy. However, the selection and combination of heuristics require careful consideration to avoid redundancy or conflicting signals that could lead to false positives or false negatives.

### **Marchal et al. (2014)**

Strengths: Development of an automated phishing detection system (PhishStorm) capable of real-time URL analysis for detecting probable phishing sites.

Provision of real-time phishing grading system to protect users from phishing material, enhancing user security.

Weaknesses: One drawback of this system is the restricted availability of data from Google Trends and Yahoo Clues. These tools offer only limited data, providing the ten most popular terms related to each requested term. Consequently, less frequently searched terms, despite their relevance to intra-URL relatedness computation. Similarly, certain less popular terms incorporated into URLs may not yield any matching results. This limitation arises because Google and Yahoo do not furnish data for terms that lack sufficient user requests, deeming them unrepresentative.

### **Baykara & Gurel, (2018)**

Strengths: Proposal of the Anti-Phishing Simulator application for providing information on phishing detection difficulty and identification of phishing emails.

Integration of Bayesian algorithm for spam email detection, enhancing the robustness of the simulator.

Weaknesses: The system's reliance on predefined rules for identifying phishing elements within emails may lead to false positives or missed detections, particularly in cases where attackers employ sophisticated tactics to bypass traditional detection methods. Moreover, the system's scalability and efficiency in handling large volumes of incoming emails may be challenged as the size of the spam database grows over time.

**Che H. et al. (2017)**

Strengths: Utilization of semantic web and occurrence matching for identifying phishing messages, leveraging semantic relationships between words and occurrences. Provision of a computation method for differentiating phishing communications, enhancing detection accuracy.

Weaknesses: Complexity of semantic web knowledge base creation and categorization information base, potentially requiring significant computational resources and expertise for implementation.

**2.2.2 Summary of Existing Technique****Table 2-1 Summary of Existing Technique**

Study	Strengths	Weaknesses	Critical Comments
Xiaoqing et al. (2018)	- Utilization of both Naive Bayes (NB) and Support Vector Machine (SVM) algorithms for classification, enhancing detection accuracy.	- Reliance on training data for generalization, which may lead to reduced performance if the data is not diverse or representative enough.	While the integration of NB and SVM algorithms enhances accuracy, the study's reliance on training data could limit its effectiveness if the dataset does not adequately represent real-world phishing scenarios.
Almseidin et al. (2019)	- Improvement of system performance through the utilization of multiple machine learning algorithms and feature selection approaches.	- Relatively small dataset size may limit generalizability. - Composition of dataset could impact findings' applicability to real-world scenarios.	Although multiple algorithms and feature selection methods were used, the study's dataset size and composition might not fully capture the complexity and diversity of real-world phishing attacks, potentially limiting the applicability of its findings.
	- Development of a heuristic technique employing Twin Support Vector	- Heuristic techniques may overlook	While the use of heuristics and TWSVM enhances detection, relying on heuristic techniques could

Rao R. S. et al. (2015)	Machine (TWSVM) classifier for detecting harmful phishing websites.	important patterns or features in data.	introduce limitations in capturing nuanced patterns in data, potentially affecting detection accuracy.
Jain & Gupta, (2017),	- Proposal of an online search tool-based approach for detecting phishing site pages with pinpoint accuracy. - Integration of heuristics with web search tool-based instrument to enhance detection accuracy.	- Potential reliance on external search engines may introduce dependency on external factors and sources. - Careful consideration needed in selecting and combining heuristics to avoid false positives or false negatives.	While the proposed approach enhances detection accuracy, reliance on external search engines and the selection of heuristics requires careful consideration to mitigate potential limitations such as dependency on external factors and the risk of false positives or negatives.
Marchal et al (2014)	- Development of an automated phishing detection system (PhishStorm) capable of real-time URL analysis. - Provision of real-time phishing grading system to protect users.	- Limited availability of data from Google Trends and Yahoo Clues may restrict the system's effectiveness.	Although PhishStorm offers real-time protection, its effectiveness might be limited by the availability of data from Google Trends and Yahoo Clues, potentially impacting its ability to detect less popular or newly emerging phishing attempts.
Baykara & Gurel, (2018)	- Proposal of the Anti-Phishing Simulator application for providing information on phishing detection difficulty and identification of phishing emails. - Integration of	- Reliance on predefined rules for identification may lead to false positives or missed detections. - Scalability and efficiency may be challenged with the growth of the spam database.	While the Anti-Phishing Simulator provides valuable information, its reliance on predefined rules and scalability challenges may affect its accuracy and efficiency, particularly in handling sophisticated phishing tactics and managing a growing spam database.



	Bayesian algorithm for spam email detection.		
Che H. et al. (2017)	- Utilization of semantic web and occurrence matching for identifying phishing messages. - Provision of a computation method for differentiating phishing communications.	- Complexity of semantic web knowledge base creation and categorization may require significant resources and expertise.	While leveraging semantic web enhances detection accuracy, the complexity of knowledge base creation and categorization could pose implementation challenges, requiring substantial resources and expertise.

### 2.3 Concluding Remark

In this comprehensive review of existing techniques on machine learning for phishing detection, a diverse array of methodologies has been explored, each presenting unique strengths and weaknesses. From leveraging Naive Bayes and Support Vector Machine algorithms for classification to heuristic techniques and online search tool-based approaches, researchers have demonstrated innovative strategies to combat phishing threats. However, challenges such as reliance on training data, scalability issues, and the complexity of knowledge base creation highlight areas for further refinement. Despite these limitations, this review underscores the robust foundation upon which future advancements in phishing detection can be built, emphasizing the importance of careful consideration and ongoing innovation in addressing evolving cyber threats.

## CHAPTER 3

### System Methodology

#### 3.1 System Development Models

The Software Development Life Cycle (SDLC) is a structured methodology used in the development of software systems. It outlines a series of phases or steps that guide the process of creating, implementing, and maintaining software, ensuring that the final product meets user needs and organizational requirements. SDLC is fundamental in software engineering as it provides a systematic approach to managing and executing software projects. It serves as a blueprint for developing software that is reliable, cost-effective, and efficient. The SDLC provides a framework that developers and project managers follow to ensure that the development process is organized, comprehensive, and predictable. It is not only about coding; rather, it encompasses everything from the initial concept to the final delivery and ongoing maintenance of the software product. There are various SDLC models, each with its own approach to software development. These models include the Waterfall Model, Agile Model, Spiral Model, and V-Model, among others. While each model has its unique characteristics, they all share a common goal: to streamline the development process and deliver a functional and reliable software product.

#### Phases of the SDLC

The phases of the SDLC are a sequence of steps that provide a roadmap for the development of a software system. These phases guide the project team through the entire life cycle of the project, ensuring that all critical aspects of development are addressed. The core phases typically include:

1. **Requirement Gathering and Analysis:** This is the initial phase of the SDLC, where the project's objectives, scope, and requirements are identified and documented. During this phase, developers, business analysts, and stakeholders collaborate to understand the needs of the end users and the business. The goal is to gather all necessary information that will guide the design and development of the system. This phase often involves conducting feasibility studies, defining project goals, and creating detailed specifications.
2. **System Design:** Once the requirements are gathered and analyzed, the next phase involves designing the system architecture. This phase includes creating

a blueprint of the system that defines its structure, components, interfaces, and data flow. The system design phase is critical as it serves as the foundation for the actual development work. It is during this phase that decisions are made regarding the choice of technologies, platforms, and tools that will be used to build the system. The design can be broken down into two sub-phases: high-level design (HLD) and low-level design (LLD). HLD focuses on the system's overall architecture, while LLD deals with the detailed design of individual components.

3. **Development (Coding):** This is the phase where the actual coding and implementation of the system take place. Developers use the design specifications created in the previous phase to write the code that brings the system to life. This phase requires careful attention to detail, as errors in coding can lead to significant issues later in the development process. The development phase may also involve integrating various components, modules, or services, ensuring that they work together as intended. Depending on the SDLC model being used, this phase can be iterative, with multiple rounds of coding, testing, and refinement.
4. **Testing:** Once the system is developed, it undergoes rigorous testing to identify and fix any bugs or issues. The testing phase ensures that the system meets the specified requirements and functions correctly under various conditions. Testing can be divided into several types, including unit testing, integration testing, system testing, and user acceptance testing (UAT). Each type of testing serves a specific purpose, from verifying individual components to validating the system's overall functionality. The goal is to ensure that the system is free of defects and performs as expected in real-world scenarios.
5. **Deployment:** After successful testing, the system is deployed to the production environment. Deployment involves installing the system on the target hardware and configuring it for operational use. This phase may also include user training, data migration, and the setup of backup and recovery systems. Deployment is a critical phase, as it marks the transition from development to production. It is essential to plan and execute the deployment carefully to minimize disruptions to business operations. Depending on the project's complexity, deployment can be done in stages (e.g., phased rollout) or as a full-scale launch.

6. **Maintenance:** The final phase of the SDLC is ongoing maintenance and support. Once the system is live, it requires regular updates, bug fixes, and enhancements to ensure it continues to meet the evolving needs of the users and the organization. Maintenance also involves monitoring the system's performance, addressing any issues that arise, and making necessary adjustments to keep the system running smoothly. This phase is critical for the long-term success of the system, as it ensures that the software remains relevant and functional over time. Maintenance activities may also include implementing new features, optimizing performance, and ensuring compliance with regulatory requirements.

### **Importance of SDLC**

The SDLC is crucial for several reasons:

- **Structured Approach:** It provides a clear and organized framework for managing the software development process, ensuring that all aspects of the project are addressed systematically.
- **Risk Mitigation:** By breaking the project into manageable phases, the SDLC helps identify and address potential risks early in the process, reducing the likelihood of project failure.
- **Quality Assurance:** The emphasis on testing and validation at each phase ensures that the final product meets the required standards of quality and reliability.
- **Cost and Time Management:** The SDLC helps project managers control costs and timelines by providing a clear plan and schedule for each phase of the project.

### **Common SDLC Models**

While the basic phases of the SDLC remain consistent across different models, the approach to each phase can vary. Some common SDLC models include:

1. **Waterfall Model:** A linear and sequential approach where each phase must be completed before the next one begins. It is simple and easy to manage but can be inflexible to changes once the project is underway.
2. **Agile Model:** An iterative and flexible approach that emphasizes collaboration, customer feedback, and continuous improvement. Agile allows for frequent releases and adaptability to changing requirements.

3. Spiral Model: Combines elements of both iterative and waterfall models, with a focus on risk assessment and reduction at each iteration or "spiral."
4. V-Model (Verification and Validation Model): A variation of the waterfall model, where each development phase is associated with a corresponding testing phase, ensuring that validation is integrated into every step of the process.

Each of these models has its advantages and disadvantages, and the choice of model depends on factors such as project size, complexity, and stakeholder requirements.

In conclusion, the SDLC is a critical framework for managing the complexities of software development. By providing a structured approach to each phase of the project, the SDLC helps ensure that the final product is of high quality, meets the needs of the users, and is delivered on time and within budget. Whether using a traditional model like Waterfall or a more flexible approach like Agile, the principles of the SDLC remain essential to the success of any software development project.

### **3.1 1 Waterfall Model**

The Waterfall Model is one of the oldest and most traditional approaches in software development, characterized by its linear and sequential structure. In this model, the development process is divided into distinct phases, and each phase must be completed before the next one begins. The process starts with the requirement gathering and analysis phase, where all the necessary system requirements are collected from the stakeholders and documented in detail. Once this phase is completed and approved, the project moves on to the system design phase, where the architecture of the system is planned and designed based on the requirements. This design phase typically involves creating system diagrams, flowcharts, and detailed design documents that serve as blueprints for the development team.

After the design phase is finalized, the project enters the implementation or coding phase, where developers write the actual code for the system. The development team follows the design specifications closely to ensure that the system is built exactly as planned. Once the coding is complete, the project transitions into the integration and testing phase, where the system is thoroughly tested for defects, bugs, and other issues. This phase is critical for ensuring that the system functions correctly and meets the specified requirements. After successful testing, the system is deployed to the

production environment in the deployment phase, making it available for end-users. Finally, the system enters the maintenance phase, where it is monitored, updated, and maintained to ensure that it continues to operate efficiently over time.

The Waterfall Model's straightforward and organized approach makes it easy to manage, especially for projects with well-defined requirements. However, its linear nature can also be a drawback, as it does not easily accommodate changes once a phase is completed. This inflexibility can be problematic if new requirements emerge or if there are changes in the project's scope during development. The model is best suited for projects where the requirements are stable and unlikely to change, such as government projects or systems with well-established functionalities.

### **3.1.2 Agile Model**

The Agile Model represents a significant departure from traditional models like Waterfall, offering a more flexible and iterative approach to software development. Agile emphasizes collaboration, customer feedback, and adaptability, making it particularly well-suited for projects where requirements are expected to change or evolve over time. Unlike the Waterfall Model, Agile does not follow a linear path; instead, it breaks the project down into smaller, manageable units called sprints. Each sprint typically lasts between one to four weeks and encompasses all phases of the development process, from requirement gathering and design to development, testing, and review. This iterative process allows the development team to produce functional software at the end of each sprint, which can be reviewed and evaluated by stakeholders.

One of the key strengths of Agile is its ability to accommodate changing requirements. Since the project is broken down into smaller cycles, any new requirements or changes can be incorporated into the next sprint without disrupting the entire development process. Agile also encourages close collaboration between developers, stakeholders, and customers, ensuring that the final product is closely aligned with user needs and expectations. This continuous feedback loop helps to catch issues early and allows for adjustments to be made quickly.

However, Agile's flexibility and iterative nature require a high level of discipline and coordination among team members. It also requires active involvement from stakeholders, who must be available to provide feedback and make decisions

throughout the development process. Additionally, because Agile focuses on delivering functional software incrementally, it can sometimes lead to scope creep, where the project continues to expand as new features are added. Despite these challenges, Agile is widely used in modern software development, particularly for projects where innovation, speed, and adaptability are crucial.

### **3.1.3 Spiral Model**

The Spiral Model is a sophisticated approach to software development that combines elements of both the Waterfall and Agile models, with a strong focus on risk management. The Spiral Model is particularly well-suited for large, complex projects where risks need to be carefully identified and mitigated. The development process in the Spiral Model is divided into several cycles or spirals, each representing a phase of the project. Each cycle begins with planning, where objectives are set, and potential risks are identified. This is followed by the risk analysis phase, where strategies are developed to mitigate or manage the identified risks.

After the risk analysis phase, the project moves into the engineering phase, where the actual development and testing of the system occur. Once the system is developed, it is evaluated by stakeholders in the evaluation phase, where feedback is gathered, and any necessary adjustments are made. This process is repeated in subsequent cycles, with each cycle refining the system further and addressing any remaining risks. The iterative nature of the Spiral Model allows for continuous refinement and adaptation of the system, ensuring that risks are managed effectively throughout the development process.

The Spiral Model is particularly advantageous for projects with significant uncertainties or risks, as it allows for thorough risk assessment and mitigation at each stage of development. However, the model's complexity and iterative nature can make it challenging to manage, especially for smaller projects. Additionally, the cost of managing and mitigating risks in each cycle can be high, making the Spiral Model more suitable for larger projects with ample resources. Despite these challenges, the Spiral Model's focus on risk management and continuous improvement makes it a valuable approach for complex and high-risk projects.

### **3.1.4 V-Model (Verification and Validation Model)**

The V-Model, also known as the Verification and Validation Model, is a structured approach to software development that extends the Waterfall Model by emphasizing the importance of testing at every stage of the development process. The V-Model is structured in a V-shape, with each development phase on one side of the "V" corresponding to a specific testing phase on the other side. For example, the requirement analysis phase is paired with user acceptance testing (UAT), ensuring that the system meets the users' needs and expectations. Similarly, the system design phase is paired with system testing, the high-level design phase with integration testing, and the low-level design phase with unit testing.

This close relationship between development and testing phases ensures that defects are identified and corrected early in the process, reducing the likelihood of major issues arising later in the project. The V-Model is particularly beneficial in projects where quality and reliability are paramount, as it provides a clear framework for validating each step of the development process. This makes it a popular choice for safety-critical systems, such as those used in the medical, automotive, and aerospace industries, where failures can have severe consequences.

However, like the Waterfall Model, the V-Model is relatively inflexible and may not be suitable for projects where requirements are expected to change frequently. Once a phase is completed, it can be difficult to go back and make changes without affecting the entire process. This rigidity can be a drawback in projects with evolving requirements or where iterative development is needed. Despite this, the V-Model's focus on verification and validation at every stage makes it a valuable tool for ensuring that the system meets all specified requirements and functions as intended.

## **3.2 Proposed Method/Approach**

The proposed method in the code is designed to detect phishing websites by leveraging a combination of feature extraction and machine learning techniques. The process begins by examining the given URL to determine if it is present in a predefined list of legitimate websites stored in a CSV file. This initial check serves as a quick filter—if the URL is found in the list, it is assumed to be legitimate, and the system returns a "0" classification, indicating that it is not a phishing website. However, if the URL is not found in the list, the system proceeds to a more detailed analysis.



The core of the method involves extracting a comprehensive set of features from the URL. These features are designed to capture various characteristics that are commonly associated with phishing websites. For instance, the method checks for the presence of an IP address in the URL, which is a common indicator of phishing attempts, as legitimate websites typically use domain names rather than raw IP addresses. Additionally, the method evaluates the length of the URL—phishing URLs often tend to be longer and more complex in an attempt to obscure their malicious intent.

Other features that are extracted include the number of slashes in the URL path (referred to as URL depth), which can indicate the structure and complexity of the website, and the presence of redirection mechanisms (such as the use of "//" within the URL). Redirection is another tactic often employed by phishing websites to lead users to malicious destinations. The method also checks if the HTTPS token is present in the domain part of the URL, as legitimate websites are more likely to use HTTPS for secure communication.

Further, the method includes checks for URL shortening services, which are often used by phishing attackers to disguise the true destination of a link. The presence of hyphens in the domain name (prefix/suffix) is another feature considered, as phishing domains often use such patterns to mimic legitimate domains. Additionally, the method evaluates domain-related features such as the availability of DNS records, the domain's web traffic ranking (using data from Alexa), the age of the domain, and the domain's expiration time. These features provide insights into the legitimacy of the website—newly created domains or those with low traffic rankings are more likely to be associated with phishing.

Beyond URL and domain-based features, the method also examines HTML and JavaScript characteristics of the website. For instance, it checks for the presence of iframe redirection, which is a common technique used by phishing websites to load malicious content without the user's knowledge. The method also inspects the effects of mouse-over events in the status bar—phishing websites often use JavaScript to change the status bar text when the user hovers over a link, misleading them about the true destination. Additionally, the method evaluates the number of web forwards (redirections), as multiple forwards can be a sign of a phishing attempt.

Once all these features are extracted, they are compiled into a feature vector, which is then passed to a pre-trained Support Vector Machine (SVM) model. This machine learning model has been trained to classify URLs based on the extracted features as

either legitimate or phishing. The model's prediction, combined with the feature analysis, determines the final classification of the URL. If the feature analysis indicates a strong likelihood of legitimacy (e.g., if most features suggest a legitimate website), the URL is classified as legitimate. Otherwise, it is classified as phishing.

In summary, the proposed method employs a comprehensive and multi-layered approach to phishing detection. By combining static checks, such as domain age and traffic, with dynamic content analysis, such as checking for iframe redirection and mouse-over effects, the system can effectively distinguish between legitimate and phishing websites. The integration of a machine learning model further enhances the accuracy of the system, making it a robust solution for phishing detection.

### 3.3 System Requirement

#### 3.3.1 Hardware

As the project undertaken is related directly to building, training, and testing an ML model, the requirements mostly fall onto the software side. Though some decent hardware capacity is required to actually support and run the model efficiently, it is not of major importance since there are alternatives to doing it locally. But as a baseline, here is a breakdown of the development and testing environment hardware –

**Table 3-1 Specifications of laptop**

<b>Description</b>	<b>Specifications</b>
Model	Asus TUF 15 series
Processor	11th Gen Intel(R) Core (TM) i7-11600H @ 2.90GHz
Operating System	Windows 10 Home Single Language
Graphic	NVIDIA GeForce GT 930MX 2GB DDR3
Memory	24GB RAM
Storage	476GB

### 3.3.2 Software

Several software tools and libraries are involved, each playing a crucial role in the development, execution, and deployment of the phishing detection and spam classification systems. These software components can be broadly categorized into programming languages, libraries, frameworks, and tools that support natural language processing (NLP), machine learning, web development, and data handling. Below is a detailed explanation of the software involved in each of the two systems.

#### **Spear Phishing Attack Detection System:**

In the spear phishing attack detection plugin, the primary software components include Python, Flask, and various Python libraries used for web scraping, natural language processing, machine learning, and web development.

**Python:** Python is the core programming language used in this system. It is chosen for its versatility, ease of use, and extensive libraries that support machine learning and web development. Python's flexibility makes it an ideal choice for implementing both the backend logic and the machine learning models required for phishing detection.

**Flask:** Flask is a lightweight web framework for Python. In this code, Flask is used to create the web application that serves as the interface for the phishing detection system. Flask handles HTTP requests, routes them to the appropriate functions, and manages the interaction between the user and the backend logic. This allows the phishing detection system to be deployed as a web service, where users can submit URLs for analysis through a web interface.

**NumPy:** NumPy is a fundamental library for numerical computing in Python. Although it is only indirectly used in this code, NumPy provides essential functions for handling arrays and performing mathematical operations, which are crucial for processing data and making predictions in machine learning models.

**Requests and BeautifulSoup:** These libraries are used for web scraping. The requests library allows the system to send HTTP requests to external websites, retrieving the HTML content needed for analysis. BeautifulSoup is a Python library used for parsing HTML and XML documents. In this code, it is used to extract specific elements from

web pages, such as web traffic data from Alexa, which is then used as a feature in the phishing detection model.

**Whois and TLDExtract:** These libraries are used for domain name extraction and WHOIS lookups. Whois is used to retrieve information about domain names, such as their creation and expiration dates, which are used to assess the legitimacy of a website. TLDExtract is a library that extracts the domain name and suffix from a URL, making it easier to analyze and compare URLs.

**String, Re, and DateTime:** These standard Python libraries are used for text processing and regular expressions (Re), which are essential for identifying patterns in URLs. The DateTime library is used for handling date and time-related operations, such as calculating domain age and expiration.

**Pickle:** The pickle module is used for serializing and deserializing Python objects. In this case, it is used to load a pre-trained machine learning model from a file (SVM\_Model.pkl). This allows the phishing detection system to quickly load the model and use it for making predictions without needing to retrain it every time.

**Flask-CORS:** This library enables Cross-Origin Resource Sharing (CORS) for Flask applications. CORS is essential when the web application needs to handle requests from different domains, ensuring that the phishing detection system can be accessed and used by various clients and services.

### **Email/SMS Spam Classification System:**

The email/sms spam classification System also relies on Python as the core programming language, but it uses a different set of libraries and tools, particularly those focused on natural language processing (NLP) and machine learning.

**Python:** Like in the spear phishing attack detection system, Python serves as the primary programming language. Its simplicity and extensive ecosystem of libraries make it ideal for developing the NLP and machine learning components of the spam classification system.

**Streamlit:** Streamlit is a popular open-source framework used for building web applications in Python, specifically for machine learning and data science projects. In this code, Streamlit is used to create an interactive user interface where users can input

## CHAPTER 3

text (such as an email or SMS) and receive a spam classification in real-time. Streamlit allows developers to build and deploy data-driven web applications with minimal effort, making it an excellent choice for this project.

**NLTK (Natural Language Toolkit):** NLTK is a comprehensive library for natural language processing in Python. It provides tools for text processing, such as tokenization, stemming, and removing stopwords. In this code, NLTK is used to preprocess the input text, transforming it into a format suitable for machine learning. This includes tokenizing the text into individual words, removing common stopwords, and stemming words to their root forms using the Porter Stemmer.

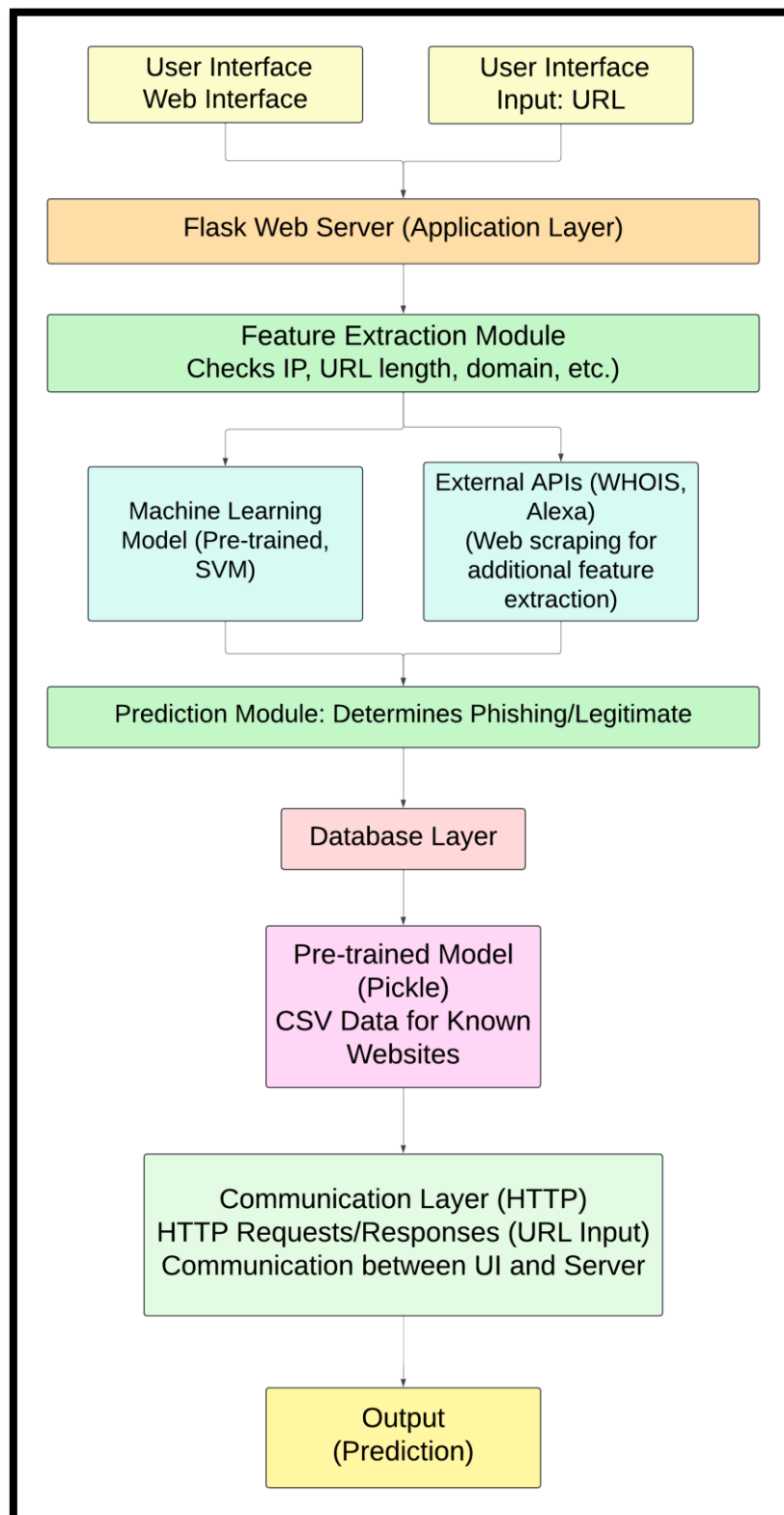
**Pickle:** Similar to the phishing detection system, pickle is used here for loading pre-trained models and vectorizers from files (`model.pkl` and `vectorizer.pkl`). This allows the spam classification system to quickly load the necessary components and make predictions without retraining the model.

**Scikit-learn (Implied using TF-IDF):** Scikit-learn is a machine learning library in Python that provides various tools for model training, evaluation, and feature extraction. In this code, the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer is used to transform the input text into a numerical format that the model can process. Scikit-learn is the library that typically provides this functionality, making it a critical component of the spam classification system.

**Numpy:** As with the phishing detection system, numpy is likely used indirectly in the background for efficient numerical computations. It supports various operations on arrays and matrices, which are essential for machine learning tasks.

In both the phishing detection and spam classification systems, Python serves as the backbone, providing the necessary flexibility and access to a rich ecosystem of libraries. Flask and Streamlit are the web frameworks that enable user interaction, allowing these systems to be deployed as web applications. Various libraries, such as NLTK, Scikit-learn, and BeautifulSoup, provide specialized functionality for natural language processing, machine learning, and web scraping, making these systems effective in their respective tasks of detecting phishing attempts and classifying spam messages. The use of pickle for model persistence ensures that these systems can quickly load and utilize pre-trained models, providing fast and accurate predictions for end-users.

### 3.4 System Architecture/Design Diagram



**Figure 3-1 System Architecture Diagram (S.P.A.D Plugin)**

## Justification

- **User Interface Layer:**

Web Interface: This layer includes the web interface where users can input a URL to be checked for phishing. The web interface is powered by Flask, a lightweight Python web framework that handles HTTP requests and responses.

User Input: The user submits a URL through a form, which is then sent to the backend for processing.

- **Application Layer:**

Flask Web Server: This layer acts as the core of the application, handling user requests and routing them to the appropriate functions. The Flask server receives the URL input from the user and passes it to the feature extraction and model prediction components.

Feature Extraction Module: This module processes the input URL to extract various features used for phishing detection. It includes different functions for checking IP addresses, URL length, presence of "@" symbols, URL depth, domain information, and more.

Machine Learning Model: The pre-trained machine learning model (loaded using pickle) is used to predict whether the URL is phishing or legitimate based on the extracted features. The model could be an SVM (Support Vector Machine) or any other classification algorithm.

- **Data Processing Layer:**

Feature Extraction: This involves parsing the URL and applying various heuristics, such as detecting the presence of IP addresses, checking the age of the domain using WHOIS data, analyzing web traffic data from Alexa, and other criteria that contribute to phishing detection.

External API Calls: The system makes external requests to gather additional data (e.g., WHOIS information, Alexa rankings) that help in determining the legitimacy of the URL.

**Prediction Module:** The prediction module uses the extracted features as input to the pre-trained model. Based on the model's output, the system decides whether the URL is phishing or legitimate.

- **Database Layer:**

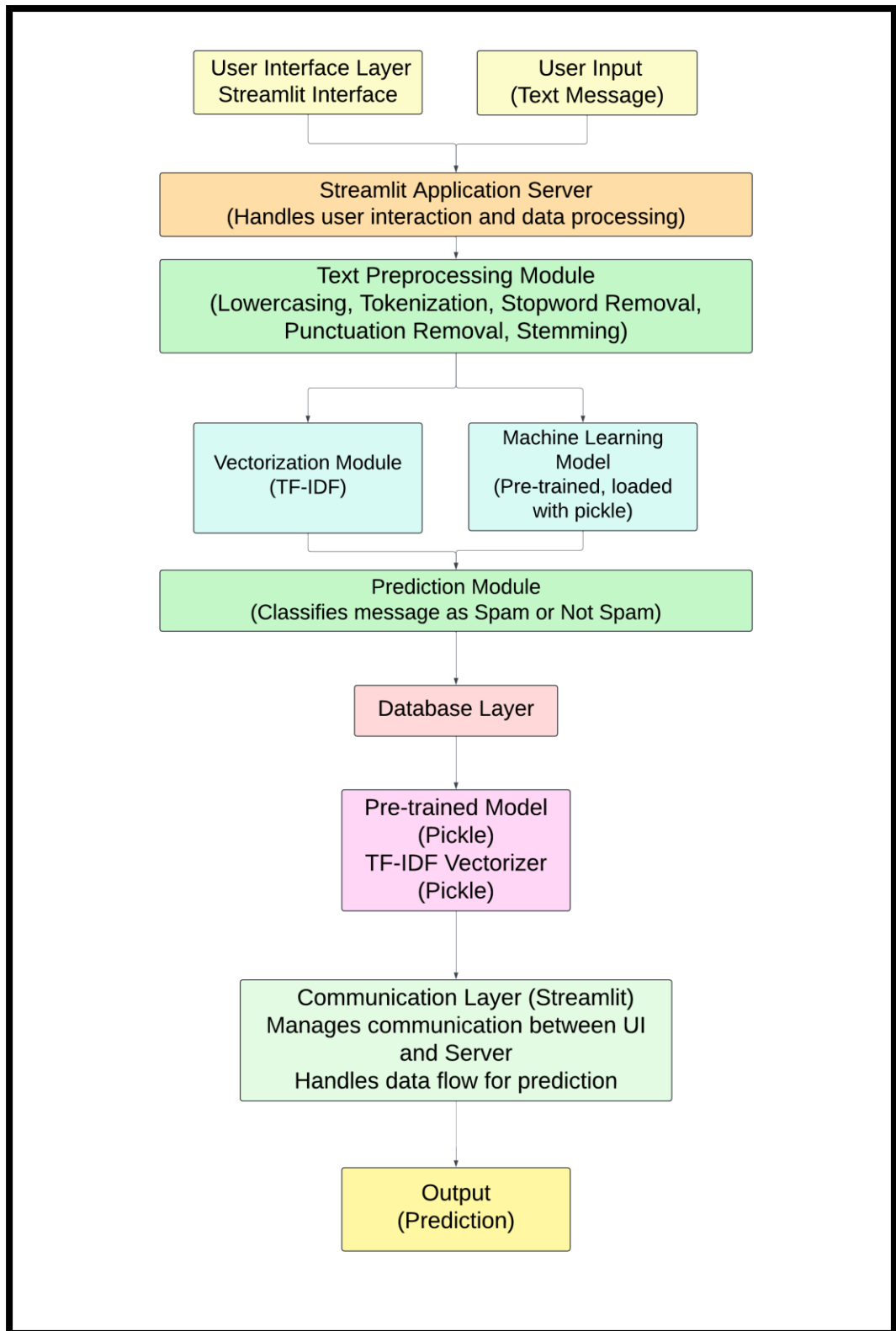
**Pre-trained Model Storage:** The machine learning model is stored in a serialized form using pickle, and it is loaded into memory when the application starts. This allows the application to use the model without retraining it every time.

**CSV File Data:** The system uses a CSV file containing a list of known legitimate websites for quick lookups. This CSV file acts as a local database for the application to check URLs against.

- **Communication Layer:**

**HTTP Requests and Responses:** Communication between the user and the server is handled through HTTP. The user submits a URL through a POST request, and the server responds with the prediction result (whether the URL is phishing or not).





**Figure 3-2 System Architecture Diagram (Email/SMS Classifier)**

## Justification

- **User Interface Layer:**

Streamlit Interface: This layer is where the user interacts with the system. The user inputs the text message (email or SMS) that they want to classify as spam or not spam. The interface is powered by Streamlit, a Python library that makes it easy to create web apps for machine learning models.

User Input: The user enters the message through a text area provided in the Streamlit interface. The input is then processed by the backend components.

- **Application Layer:**

Streamlit Application Server: This layer is responsible for handling user interactions and managing the flow of data between the user interface and the backend components. When the user submits a message, the Streamlit server processes the input, triggers the text preprocessing, and passes the processed data to the model for prediction.

Text Preprocessing Module: This module preprocesses the input text by transforming it into a format that the machine learning model can understand. It performs tasks such as lowercasing, removing stopwords, punctuation, and stemming using NLTK (Natural Language Toolkit).

Machine Learning Model: The pre-trained machine learning model (loaded using pickle) is used to classify the input message as either spam or not spam. The model is likely trained on a dataset of labeled messages (spam and non-spam) and utilizes techniques like TF-IDF for vectorization and a classifier like Naive Bayes, SVM, or another algorithm.

- **Data Processing Layer:**

Text Preprocessing: The text preprocessing module cleans and prepares the input message by applying several steps, such as tokenization, removal of stopwords, and stemming. This step ensures that the input data is in the correct format for the model.

Vectorization Module: The preprocessed text is then converted into a numerical format using a vectorizer, such as TF-IDF (Term Frequency-

Inverse Document Frequency). This step transforms the text data into a feature vector that the model can use for prediction.

- **Model Prediction Layer:**

**Prediction Module:** The processed feature vector is fed into the pre-trained machine learning model. The model predicts whether the message is spam or not based on the input features.

**Prediction Output:** The result of the prediction (spam or not spam) is then displayed to the user through the Streamlit interface.

- **Database Layer:**

**Pre-trained Model Storage:** The pre-trained model is stored in a serialized form using pickle. The model is loaded into memory when the application starts, enabling the system to use the model without retraining it every time.

**TF-IDF Vectorizer:** The TF-IDF vectorizer used for converting the text into numerical form is also stored using pickle and loaded into memory during runtime.

- **Communication Layer:**

**Streamlit Communication:** Communication between the user and the backend is handled within the Streamlit framework. The user submits the message, and the server processes it, returning the prediction result.

### **3.5 Expected Challenges**

The project is expected to encounter several challenges, including data quality and quantity issues in obtaining labelled datasets for training machine learning models. Additionally, selecting relevant features for effective feature engineering poses a complex task that requires domain knowledge and experimentation to avoid overfitting. Achieving high performance in detecting spear phishing attacks may be challenging, as models may struggle to generalize well to unseen data, leading to false positives or false negatives. Moreover, the threat of adversarial attacks presents a significant concern, as malicious actors may actively attempt to evade detection by crafting sophisticated attacks. Ensuring scalability to handle large volumes of data in real-time while maintaining detection accuracy is another non-trivial task. Moreover, ethical concerns regarding privacy and data protection must be diligently attended to throughout the project, particularly given the examination of potentially sensitive data such as email content and user behaviour. Overcoming these challenges will require a multidisciplinary approach, continuous iteration, experimentation, and collaboration across cybersecurity, machine learning, and software engineering domains.

### 3.6 Project Milestone

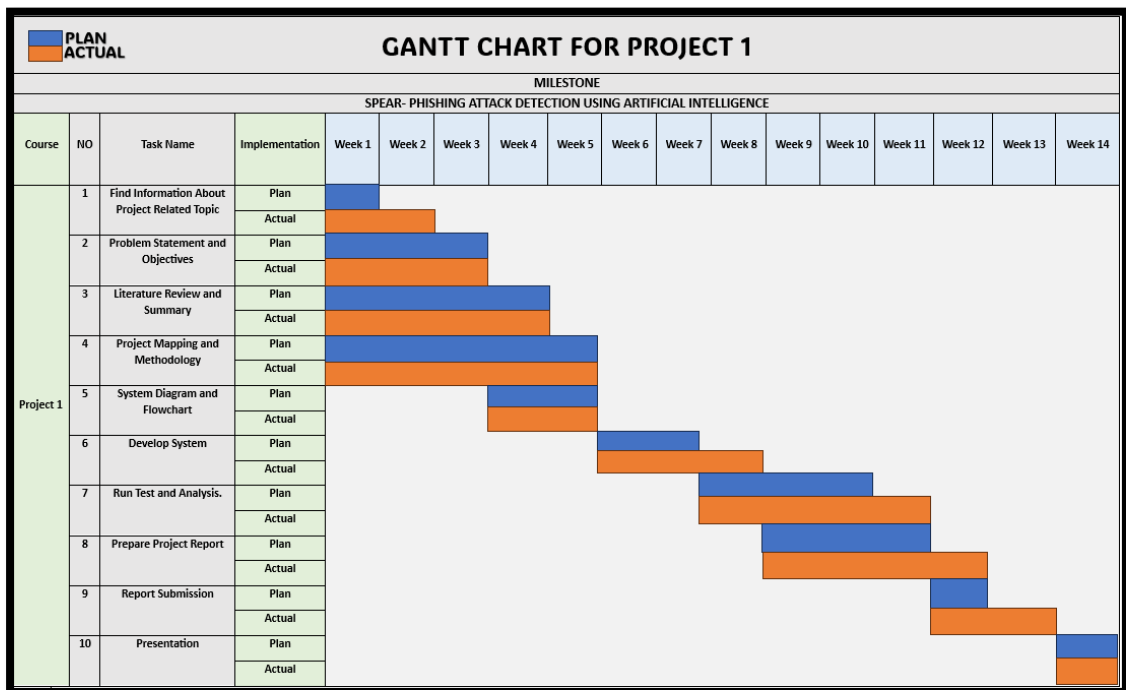


Figure 3-3 The Gantt Chart for FYP1

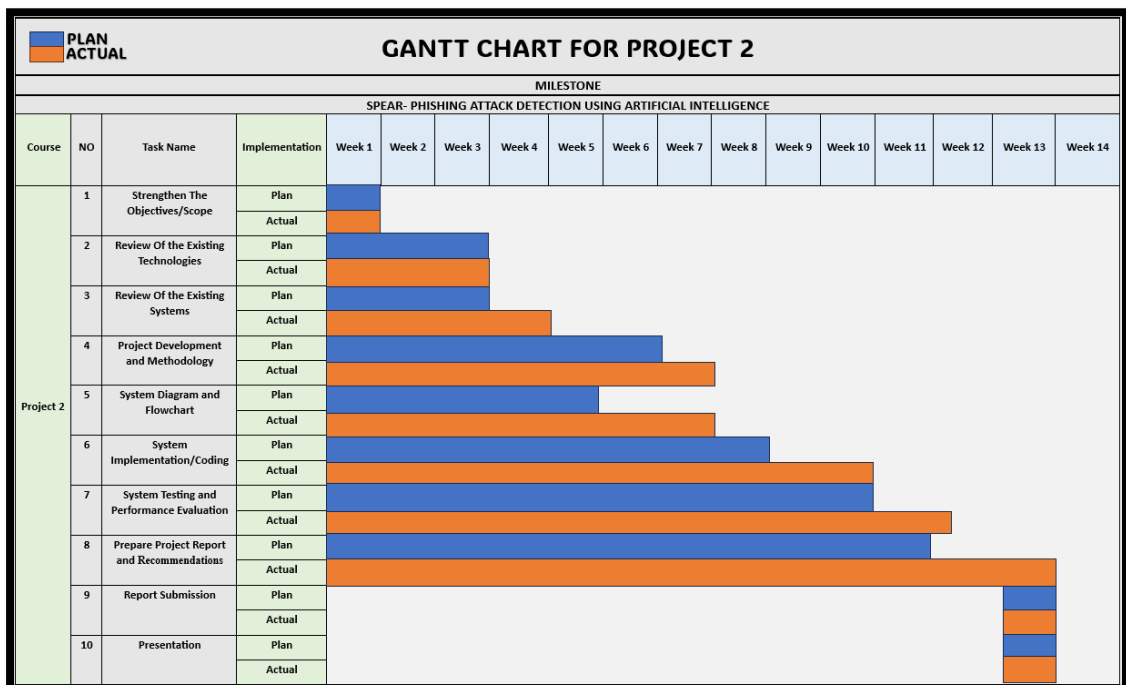


Figure 3-4 The Gantt Chart for FYP2

### **3.7 Concluding Remarks**

In this chapter, the methodology and software used for system development and analysis have been thoroughly examined. The software development approach utilized includes both traditional and modern methodologies, such as the Waterfall model for its linear, sequential phases and the Agile model for its iterative and flexible nature. The choice of these models aligns with the project's need for structured yet adaptable processes. Furthermore, the chapter highlights the specific tools and software employed, including Flask for building the web server, Scikit-Learn for machine learning, and Streamlit for the user interface, each contributing to the robustness and efficiency of the system. The integration of these tools into a coherent methodology ensures that the development process is both systematic and responsive to changes, ultimately leading to a well-rounded and functional system. This comprehensive approach underscores the importance of selecting appropriate methodologies and software to achieve project goals effectively and adapt to evolving requirements.

4.1 System Architecture

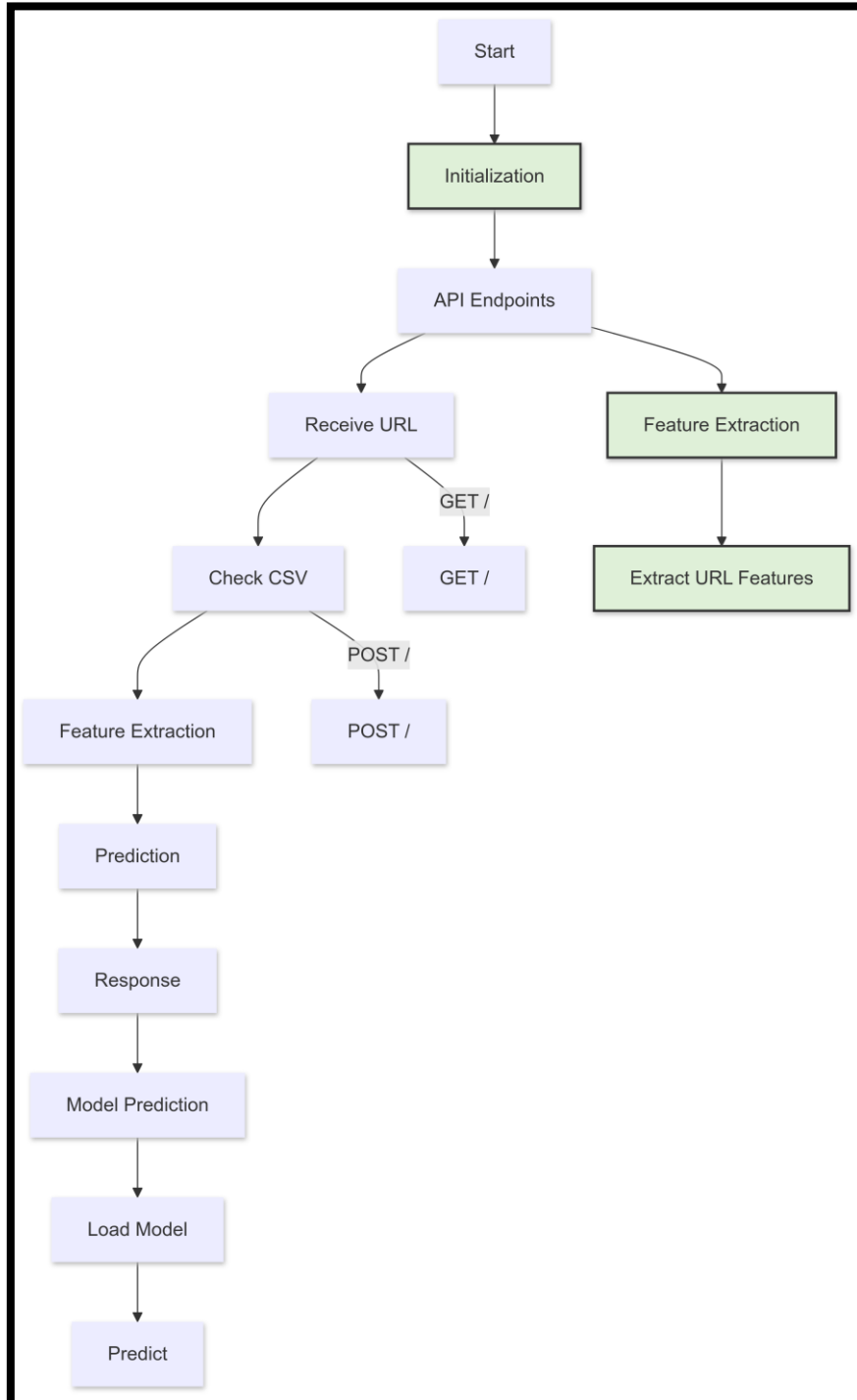
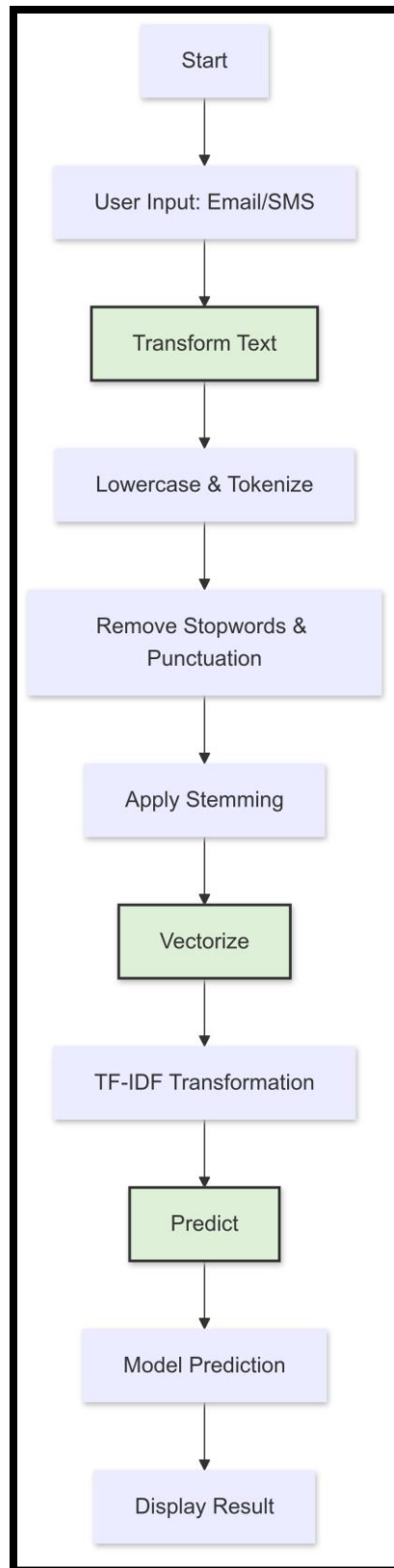


Figure 4-1 Block Diagram (S.P.A.D Plugin)



**Figure 4-2 Block Diagram (Email/SMS Classifier)**



## 4.2 Functional Modules in the System

### Functional Modules (S.P.A.D Plugin)

**1. Initialization and Imports:** The first code initializes by importing various libraries and modules needed for processing URLs and making predictions. These include numpy, flask for web framework functionalities, pickle for loading pre-trained models, whois for domain information, tldextract for domain extraction, and other libraries for handling URLs, requests, and HTML parsing.

**2. Feature Extraction Functions:** This section defines functions to extract features from URLs for phishing detection:

- **havingIP(url):** Checks if the URL contains an IP address instead of a domain name.
- **haveAtSign(url):** Detects the presence of special characters like @ in the URL.
- **getLength(url):** Determines if the URL length is suspiciously long.
- **getDepth(url):** Counts the number of slashes in the URL path to determine its depth.
- **redirection(url):** Checks for redundant redirection symbols (//) in the URL.
- **httpDomain(url):** Verifies if 'https' is present in the domain part of the URL.
- **tinyURL(url):** Identifies if the URL uses known URL shortening services.
- **prefixSuffix(url):** Checks for hyphens in the domain which might indicate phishing.

**3. Domain-Based and HTML/JavaScript Feature Extraction:** Additional functions are used to extract features related to the domain and webpage:

- **web\_traffic(url):** Uses Alexa rankings to assess web traffic and infer the legitimacy of the site.
- **domainAge(url):** Determines the domain age to infer if it's a newly registered potentially phishing domain.
- **domainEnd(domain\_name):** Checks the expiration date of the domain to assess its credibility.
- **iframe(response), mouseOver(response), forwarding(response):** Analyze the webpage content for indicators like iframes, mouse-over scripts, or excessive redirection which could suggest phishing.

**4. CSV Checking and Prediction Logic:** The code includes functionality to check if a URL is listed in a CSV file of known malicious sites:

- **checkCSV(url):** Checks if the URL is in a list of known malicious websites.
- **featureExtraction(url):** Aggregates all feature extraction functions to prepare the feature set for prediction.
- **predict():** Handles the prediction request, processes the URL, extracts features, and classifies the URL using the pre-trained model.

#### **5. Flask Web Application:**

- **/ Route:** A simple route that returns "Hello World."
- **/post Route:** Handles POST requests for URL prediction. It extracts features, checks the URL against known data, and returns the classification result.

### **Functional Modules (Email/SMS Classifier)**

**1. Initialization and Imports:** The second code begins by importing necessary libraries for building a Streamlit web application and processing text data. Libraries include streamlit for creating the web interface, pickle for loading pre-trained models and vectorizers, string and nltk for text preprocessing tasks.

#### **2. Text Transformation Function:**

- **transform\_text(text):** This function processes the input text to prepare it for classification:
  - **Lowercasing and Tokenization:** Converts text to lowercase and tokenizes it into words.
  - **Removal of Non-Alphanumeric Tokens:** Removes any non-alphanumeric tokens.
  - **Stopwords and Punctuation Removal:** Eliminates common stopwords and punctuation.
  - **Stemming:** Applies the Porter Stemmer to reduce words to their root forms.

### 3. Model and Vectorizer Loading:

- **tfidf = pickle.load(open('vectorizer.pkl','rb')):** Loads the TF-IDF vectorizer used to transform text into numerical features.
- **model = pickle.load(open('model.pkl','rb')):** Loads the pre-trained spam classification model.

### 4. Streamlit Web Application:

- **Title and Input:** Sets up the Streamlit app's title and text area for user input.
- **Predict Button:** When the user clicks the "Predict" button, the following sequence occurs:
  - **Text Preprocessing:** Calls transform\_text to clean and process the input SMS.
  - **Vectorization:** Transforms the processed text into a feature vector using the TF-IDF vectorizer.
  - **Prediction:** Uses the pre-trained model to predict if the SMS is spam or not.
  - **Display Result:** Displays the prediction result as "Spam" or "Not Spam" on the web page.

Each module is designed to handle specific tasks related to processing, analyzing, and classifying text data, either for phishing detection or spam classification, and provides a user-friendly interface for interaction.

4.3 System Flowchart

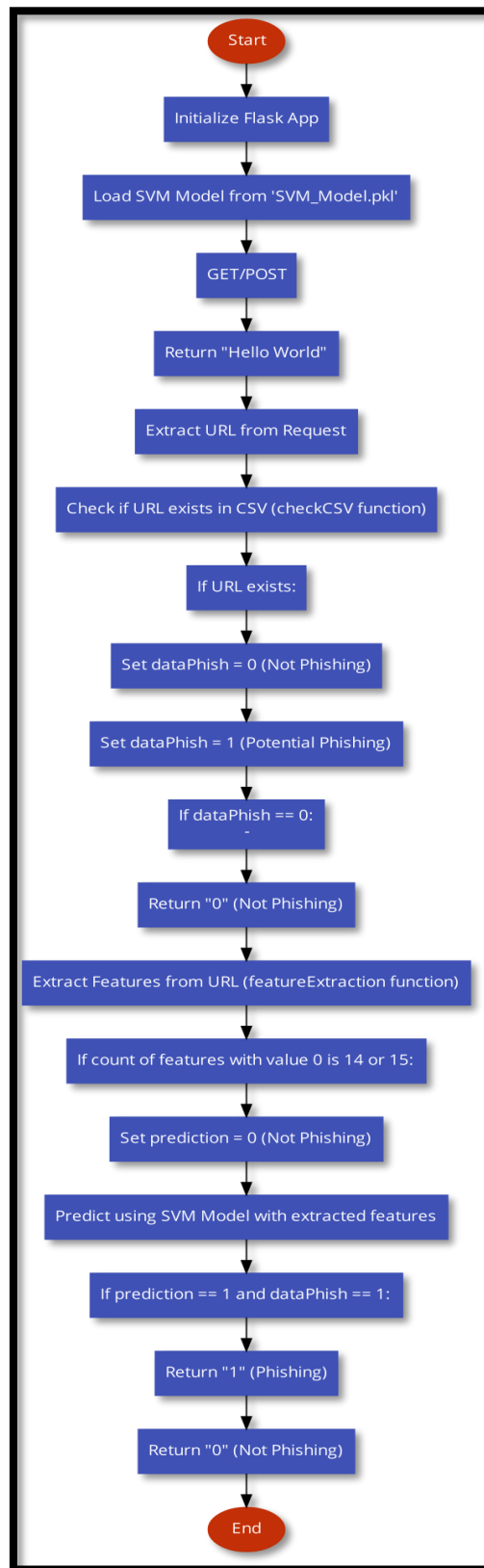


Figure 4-3 System Flowchart (S.P.A.D Plugin)

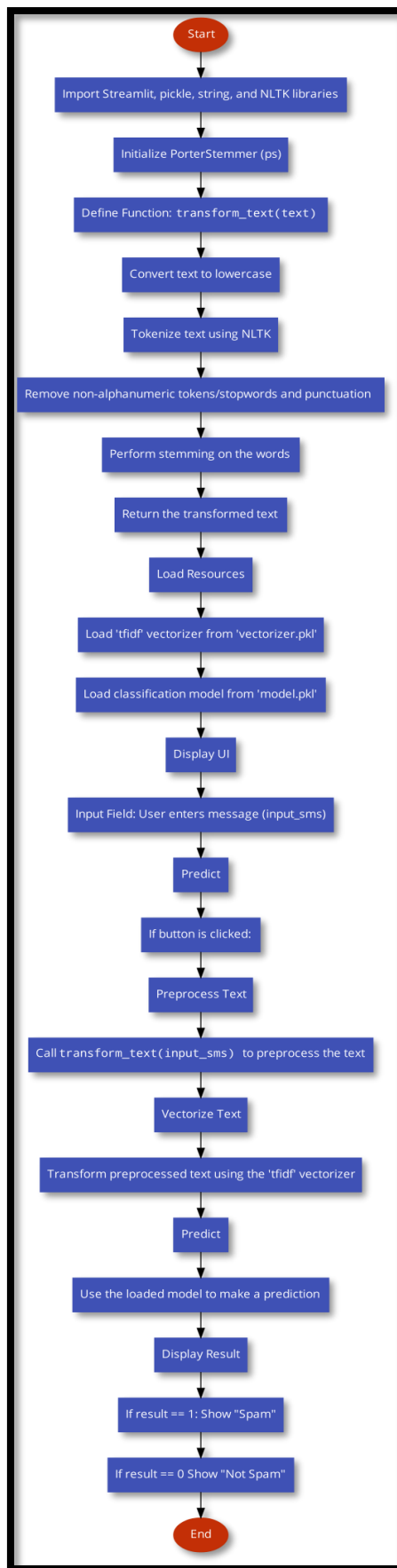


Figure 4-4 System Flowchart (Email\SMS Classifier)

### 4.3.1 System Flowchart (S.P.A.D Plugin)

The S.P.A.D Plugin system is designed to detect phishing URLs by analyzing various features of a given URL.

1. **Initialization:** The application starts by initializing a Flask app and loading a pre-trained Support Vector Machine (SVM) model from a pickle file (SVM\_Model.pkl). This model is used later to make predictions about whether a URL is phishing or legitimate.
2. **Home Route (/):** When a GET or POST request is made to the root URL (/), the application returns a simple "Hello World" message. This route is mainly for testing or ensuring that the server is running.
3. **Prediction Route (/post):** This is the main route for detecting phishing URLs. When a POST request is made to this route, the application extracts the URL from the request's form data. It then checks if the URL exists in a predefined CSV file of popular websites using the checkCSV function. If the URL is found in the CSV, it is marked as not phishing (dataPhish=0), and the application immediately returns "0" to indicate that the URL is safe.

If the URL is not found in the CSV, the application sets dataPhish=1, indicating potential phishing. The application then extracts various features from the URL using the featureExtraction function. This function checks for characteristics such as the presence of an IP address in the URL, the use of "@" symbols, URL length, depth, and several others related to domain properties and HTML content.

Based on the extracted features, the application checks if the count of features with a value of 0 is 14 or 15, which would indicate that the URL is likely safe. If so, the application sets prediction=0 and returns "1" to indicate the URL is legitimate. Otherwise, the SVM model is used to predict whether the URL is phishing. If the model predicts phishing and the URL was not found in the CSV (dataPhish=1), the application returns "1" to indicate phishing. If the prediction and dataPhish values do not indicate phishing, the application returns "0" to indicate a safe URL.

### 4.3.2 System Flowchart (Email\SMS Classifier)

This Email\SMS Classifier system allows users to input an email or SMS message and determine whether it is spam or not.

1. **Initialization:** The application starts by loading a pre-trained model and a TF-IDF vectorizer from pickle files (vectorizer.pkl and model.pkl). The TF-IDF vectorizer is used to convert the input text into numerical features that the model can understand.
2. **User Interface:** The application has a simple user interface created with Streamlit. It displays a title ("Email/SMS Spam Classifier") and provides a text area where users can enter the message they want to classify.
3. **Text Transformation:** When the user clicks the "Predict" button, the application preprocesses the input text using the transform text function. This function performs several steps:
  - It converts the text to lowercase.
  - Tokenizes the text into individual words.
  - Filters out non-alphanumeric characters and stopwords (common words like "the", "is", etc.).
  - Applies stemming using the PorterStemmer, reducing words to their root forms (e.g., "running" to "run"). The transformed text is then converted into a numerical vector using the TF-IDF vectorizer.
4. **Prediction:** The vectorized input is fed into the pre-trained model, which predicts whether the message is spam (1) or not spam (0).
5. **Display Results:** Finally, the application displays the result to the user. If the model predicts the message as spam, the application shows a "Spam" header. Otherwise, it displays "Not Spam."

S.P.A.D system focuses on URL phishing detection using a machine learning model and feature extraction from the URL. The app checks whether the URL is already known, or extracts features to predict its legitimacy using the SVM model. Email/SMS Classifier meanwhile provides a user-friendly interface for detecting spam in messages using a machine learning model. It preprocesses the text, converts it into numerical data using TF-IDF, and classifies it as spam or not using a pre-trained model.

### **4.4 Concluding Remark**

Chapter 4 has provided a comprehensive and detailed exploration of the system design, establishing a solid foundation for the successful implementation of the proposed solution. This chapter represents a critical step in translating the theoretical concepts and problem definitions discussed in earlier sections into a practical, executable system that can effectively address the identified challenges.

The chapter began by presenting the overall system architecture, carefully outlining the various components and their interactions. The architecture was designed with a focus on modularity, scalability, and efficiency, ensuring that the system can handle the expected workload while remaining adaptable to future enhancements. Key components such as the machine learning model, feature extraction techniques, user interface, and the backend infrastructure were discussed in depth, highlighting their roles and how they integrate to form a cohesive solution. In addition to the architectural overview, the chapter delved into the design of specific system components. The process of feature extraction, which forms the backbone of the system's ability to differentiate between legitimate and phishing URLs, was explained step by step. By selecting and extracting relevant features from URLs, the system can make informed predictions using the trained machine learning model. This careful attention to feature engineering is crucial for maximizing the model's accuracy and minimizing false positives and negatives.

The use of flowcharts in this chapter served to visually represent the logical flow of the system, making it easier to understand how each component interacts with the others. The flowcharts provided clarity on the decision-making processes within the system, particularly in how the model's predictions are generated and how the system responds to user inputs. These visual tools are invaluable for both developers and stakeholders, ensuring that the system's design is transparent and accessible.

The chapter also addressed potential challenges and considerations in the system design, such as the handling of edge cases and the need for continuous model updates. By anticipating these challenges, the design is better equipped to maintain its effectiveness over time, even as new phishing techniques emerge.



## CHAPTER 5

### SYSTEM IMPLEMENTATION

#### 5.1 Software Setup

##### S.P.A.D PLUGIN APPLICATION

This Flask-based web application is designed to predict whether a given URL is phishing or legitimate by examining various characteristics of the URL and its associated domain. The application uses a Support Vector Machine (SVM) model to make predictions, which is pre-trained and loaded using the Python pickle library. The goal of this application is to enhance online safety by automatically identifying potentially malicious URLs.

The application leverages several libraries. **Flask** is used to create the web API that handles incoming requests and responses. **Flask-CORS** is integrated to manage cross-origin resource sharing (CORS) and allows the API to accept requests from different origins, which is essential for web-based applications that may interact with clients hosted on various domains. **Pickle** is used to load the trained SVM model that will perform the final URL classification. Other important libraries include **BeautifulSoup** and **requests**, which help retrieve and parse web content. These libraries are particularly useful for extracting website traffic data from third-party services such as Alexa. Additionally, **tdextract** and **whois** are used to analyze the domain, extracting detailed information about the domain's ownership, creation, and expiration dates.

For feature extraction, libraries like **Numpy** are used for handling numerical data, and the **datetime** and **dateutil** libraries are used to handle time-based features, such as the age of the domain. Regular expressions, via the **re** library, are utilized to match patterns within the URL, helping detect shortened URLs or malicious script tags. Collectively, these libraries provide the tools needed to analyze various URL characteristics and make an informed prediction.

- **URL Feature Extraction and Analysis**

The application focuses on extracting a wide range of features from the URL to assess its potential risk. The features are divided into three main categories: address-bar-based, domain-based, and HTML/JavaScript-based features.

- Address-bar-based features analyze the structure of the URL itself. For example, **havingIP** checks whether the URL contains an IP address instead of a domain name, which can often indicate a phishing attempt. Similarly, **haveAtSign** checks for the presence of the "@" symbol, often used in phishing URLs to mislead users. The **getLength** function determines if the URL is suspiciously long, which may signal an attempt to obfuscate the true destination of the link. Another key feature, **redirection**, checks for multiple instances of redirection in the URL, specifically the use of "///" beyond the typical scheme (e.g., https://). The application also checks for the presence of HTTPS in the domain via **httpDomain**, although some phishing sites may still use HTTPS.
- Domain-based features are extracted to assess the trustworthiness of the domain. **web\_traffic** uses Alexa rankings to measure how much traffic the domain receives. Low traffic can indicate that the domain is not widely visited, raising suspicion. The age of the domain is also considered using **domainAge**, with recently registered domains being more likely to be associated with phishing. Additionally, **domainEnd** looks at how soon the domain registration will expire, as short-lived domains are often associated with phishing activities.
- HTML and JavaScript-based features are extracted to detect malicious behavior on the webpage itself. For instance, **iframe** examines whether the page contains suspicious iframe redirections, which can be used to load malicious content. **mouseover** checks for malicious scripts that trigger actions when the user hovers over an element on the webpage. The **forwarding** function determines the number of redirects, with multiple redirects being another potential phishing indicator.
- **Additional Features and Checks**

One unique feature of this application is the **checkCSV** function, which cross-references the URL with a CSV file of previously scraped websites. This CSV likely contains URLs from reputable sources, so if the URL exists in this file, it is considered legitimate. This adds an extra layer of verification, especially for popular websites.

- **Prediction Logic and Model**

The main endpoint of the application is `/post`, which accepts a POST request with a URL in the request body. When a URL is submitted, the application first checks it against the CSV file of known websites. If the URL is found in the CSV, the application immediately returns a response indicating the URL is legitimate. If the URL is not found in the CSV, the application proceeds to extract the various features outlined earlier. These features are then passed into the pre-trained SVM model, which classifies the URL as either phishing or legitimate.

- The result is returned to the user as one of three possible outcomes: "0" for legitimate, "-1" for phishing, or "1" for uncertain cases where the URL may be flagged by the model but is not definitively categorized by the CSV check. The use of an SVM model ensures that the prediction is based on well-established machine learning algorithms, which are particularly adept at binary classification tasks like this.
- This application serves as an effective tool for identifying potentially dangerous URLs. By extracting features from the URL and performing additional checks, such as cross-referencing with a CSV of known domains, it can reliably determine whether a given URL poses a phishing risk. The modular design, relying on a mix of web scraping, domain analysis, and machine learning, makes the application robust and adaptable to new threats. Ultimately, it is a practical solution for helping users stay safe online by identifying malicious URLs before they become a problem.

### **EMAIL\SMS CLASSIFIER APPLICATION**

This application relies on several important libraries. **Streamlit** is used to create the web interface, where users can input a message and view the prediction. **NLTK** (Natural Language Toolkit) is used for text preprocessing, including tokenization, stopword removal, and stemming. The `nlk.corpus` module provides access to a list of stopwords, which are common words (like "the", "and", etc.) that are generally removed in text classification tasks. **Pickle** is used to load the pre-trained machine learning model and the TF-IDF (Term Frequency-Inverse

Document Frequency) vectorizer, which is responsible for transforming the input text into a format that the machine learning model can understand.

- **Preprocessing the Text**

- ❖ The function `transform_text` is a key part of the application and is responsible for preprocessing the user's input message. Preprocessing is crucial in natural language processing (NLP) because it helps standardize and clean the text before it's passed into the machine learning model.
- ❖ The first step in `transform_text` is to **convert the text to lowercase**. This ensures that words like "Hello" and "hello" are treated the same way by the model.
- ❖ Next, **tokenization** is performed using `nltk.word_tokenize()`, which breaks the text into individual words or "tokens." Once the text is tokenized, the code removes any tokens that are not alphanumeric using the `isalnum()` method, keeping only words and numbers.
- ❖ The function then **removes stopwords** and **punctuation**. Stopwords are common words that typically do not contribute much meaning to a classification task. NLTK provides a pre-built list of English stopwords, which is used to filter out these words. Similarly, punctuation is removed because it doesn't help in determining whether the text is spam or not.
- ❖ Finally, the function applies **stemming** using `PorterStemmer` from NLTK. Stemming reduces each word to its root form, so that words like "running" and "runs" are both converted to "run." This reduces the dimensionality of the input and helps the model generalize better.
- ❖ The transformed text is returned as a single string, with words joined by spaces, and is ready to be vectorized.

- **Vectorization and Prediction**

- ❖ Once the text has been transformed, it is passed to the TF-IDF vectorizer that has been loaded using pickle. **TF-IDF** is a technique used to convert a text document into numerical features, which the machine learning model can understand. The vectorizer was likely trained on a dataset of spam and non-spam messages to learn important features for classification.

- ❖ The preprocessed text is transformed into a numerical vector using the command `tfidf.transform([transformed_sms])`. The vectorized input is then passed into the machine learning model for prediction. The model's output is a binary result, where 1 typically indicates spam, and 0 indicates not spam.
- **Displaying the Result**
  - ❖ Finally, the result of the prediction is displayed to the user in the Streamlit interface. Depending on the prediction made by the model, the application will display either "Spam" or "Not Spam" as the output. This simple interface allows users to easily classify messages by typing them into the text box and clicking the "Predict" button.

In summary, this application takes an input message, preprocesses it to remove irrelevant features like stopwords and punctuation, and converts it into a numerical vector using a pre-trained TF-IDF vectorizer. The vector is then passed into a pre-trained machine learning model, which classifies the message as either spam or not spam. The entire process is wrapped into an interactive web interface built using Streamlit, making it user-friendly and easy to use for real-time spam classification.

## **5.2 SETTING AND CONFIGURATION**

### **S.P.A.D PLUGIN APPLICATION**

#### **1. Import Libraries**

Imported essential libraries for the project, including numpy for numerical operations, Flask for creating the web application, and pickle for loading the pre-trained machine learning model. Used requests for making HTTP requests, BeautifulSoup for parsing HTML content, and whois for retrieving domain information. Other libraries such as re (for regular expressions), string (for string operations), and tldextract (for extracting domain information) were also included to support various features of the application.

## 2. Load the Pre-trained Model

Loaded the Support Vector Machine (SVM) model from a file named SVM\_Model.pkl using the pickle library. This model is used to predict whether a URL is phishing or legitimate based on extracted features.

## 3. Feature Extraction Functions

Several functions were defined to extract features from the URLs:

- **havingIP** checks if an IP address is present in the URL.
- **haveAtSign** detects the presence of special characters like @.
- **getLength** determines the length of the URL.
- **getDepth** calculates the number of slashes in the URL path.
- **httpDomain** checks if the domain part of the URL contains 'https'.
- **tinyURL** identifies URL shortening services.
- **prefixSuffix** detects hyphens in the domain part of the URL.
- **web\_traffic** evaluates website traffic using Alexa rankings.
- **domainAge** estimates the domain's age based on its creation date.
- **domainEnd** assesses the domain's expiration date.
- **iframe** looks for iframe tags in the HTML response to detect potential redirections.
- **mouseOver** checks for JavaScript mouseover events.
- **forwarding** counts the number of HTTP redirects.
- **checkCSV** verifies if the URL exists in a list of known phishing or legitimate sites.

## 4. Feature Extraction

The featureExtraction function consolidates all the individual feature extraction functions. It creates a feature list for each URL, which includes attributes like the presence of IP addresses, special characters, URL length, domain information, and HTML/JavaScript content.

## 5. Application Routes

Setting up two routes in the Flask application:

The root route (/) simply returns a "Hello World" message, confirming that the server is running.

The /post route handles POST requests, where it receives the URL, processes it to extract features, and uses the SVM model to predict whether the URL is phishing. The prediction result is then returned based on the model's output and additional checks against known phishing sites.

### **6. Handle Requests and Predictions**

In the /post route, after extracting features from the URL, the application checks if the URL is listed in the CSV file of known phishing sites. If it is, the application returns a result indicating a higher likelihood of phishing. If not, the feature list is passed to the SVM model for prediction, and the result is returned accordingly.

### **7. Run the Application**

The Flask application is started with `app.run(debug=True)`, which launches the server in debug mode to facilitate development and testing.

## **EMAIL\SMS CLASSIFIER APPLICATION**

### **1. Import Libraries**

- `import numpy as np`: Imports the NumPy library for handling arrays and numerical operations.
- `import pandas as pd`: Imports Pandas for data manipulation, especially for handling DataFrames.

### **2. Loading Data**

- `df = pd.read_csv('spam.csv')`: Reads the spam.csv file into a Pandas DataFrame `df`.
- `df.sample(5)`: Displays 5 random samples from the DataFrame to get a sense of the data.
- `df.shape`: Displays the dimensions of the dataset (rows, columns).
- `df.info()`: Provides a concise summary of the DataFrame, including the column names, non-null counts, and data types.

### **3. Cleaning the Data**

- `df.drop()` (Removes unnecessary columns, which may contain irrelevant or empty data)

- `df.rename()`(Renames columns to more meaningful names) The 'v1' column becomes 'target' (the label) and 'v2' becomes 'text' (the message content).

### 4. Label Encoding

- `from sklearn.preprocessing import LabelEncoder`: Imports LabelEncoder for converting categorical labels (spam/ham) to numerical values.
- `df['target']` :Encodes the 'target' column into binary values: 0 for "ham" and 1 for "spam".

### 5. Handling Missing/Duplicate Data

- `df.isnull().sum()`: Checks for missing values in each column.
- `df.duplicated().sum()`: Checks for duplicate rows in the DataFrame.
- `df = df.drop_duplicates(keep='first')`: Removes duplicate rows, retaining the first occurrence.

### 6. Visualizing Data

- `import matplotlib.pyplot as plt`: Imports Matplotlib for visualization.
- `plt.pie(df['target'].value_counts(), labels=['ham', 'spam'], autopct="%0.2f")`: Creates a pie chart to show the distribution of "ham" and "spam".

### 7. Feature Engineering

- `df['num_characters'] = df['text'].apply(len)`: Adds a column representing the number of characters in each message.
- `df['num_words'] = df['text'].apply(lambda x: len(nltk.word_tokenize(x)))`: Adds a column for the number of words in each message.
- `df['num_sentences'] = df['text'].apply(lambda x: len(nltk.sent_tokenize(x)))`: Adds a column for the number of sentences.

### 8. Data Description

- `df[['num_characters', 'num_words', 'num_sentences']].describe()`: Provides summary statistics for the new columns.
- `df[df['target'] == 0]...`: Describes the statistics of 'ham' messages.
- `df[df['target'] == 1]...`: Describes the statistics of 'spam' messages.



## 9. Data Visualization

- `import seaborn as sns`: Imports Seaborn for advanced visualizations.
- Various `sns.histplot(...)` commands: Creates histograms to compare distributions of `num_characters`, `num_words` for "ham" and "spam" messages.

## 10. Text Preprocessing

- The text is preprocessed to clean and prepare it for model training. This includes:
  - a. **Lowercasing**: Converts all text to lowercase.
  - b. **Tokenization**: Splits the text into words.
  - c. **Removing Stop Words & Punctuation**: Removes common English stop words and punctuation.
  - d. **Stemming**: Reduces words to their root forms using the PorterStemmer.

## 11. WordCloud Visualization

- `from wordcloud import WordCloud`: Generates a word cloud to visualize frequently occurring words in "spam" and "ham" messages.
- `spam_wc = wc.generate(...)`: Creates a word cloud for spam messages.
- `ham_wc = wc.generate(...)`: Creates a word cloud for ham messages.

## 12. Text Vectorization

- Converts text into numerical vectors using either **Bag of Words** or **TF-IDF** techniques.
- `from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer`: Imports vectorizers.
- `X = tfidf.fit_transform(df['transformed_text']).toarray()`: Transforms the cleaned text into numerical features using TF-IDF.

## 13. Model Building

- Splits the data into training and testing sets: `X_train, X_test, y_train, y_test = train_test_split`.
- Uses several classifiers from scikit-learn:
  - ❖ Naive Bayes (`GaussianNB`, `MultinomialNB`, `BernoulliNB`).
  - ❖ Logistic Regression, `SVC`, Decision Trees, `RandomForest`, `KNeighborsClassifier`.

- `train_classifier(...)`: A helper function that trains a given classifier and returns its accuracy and precision.

### 14. Performance Comparison

- Models are trained and evaluated using `accuracy_score` and `precision_score`.
- The results are stored in a DataFrame `performance_df` and visualized using bar plots.

### 15. Ensemble Techniques

- **Voting Classifier**: Combines predictions from multiple models (SVM, Naive Bayes, Extra Trees).
- **Stacking Classifier**: Uses one model to aggregate predictions from multiple base models.

### 16. Model Saving

- The trained model and vectorizer are saved using pickle for future use.

This entire process is a typical workflow for text classification, involving data loading, cleaning, feature extraction, model building, evaluation, and optimization.

## 5.3 System Operation

### User Experience with the Plugin

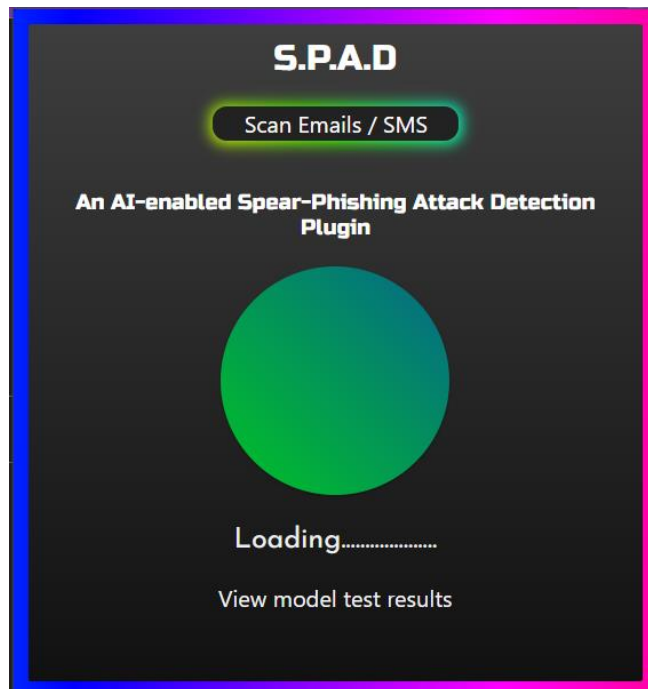
The plugin interface starts with a prominent header labeled "S.P.A.D," which stands for Spear-Phishing Attack Detection. This header clearly identifies the purpose of the plugin to the users. Below this header, there is a button titled "Scan Emails / SMS." This button provides a link to Email\SMS Classifier Application service, where users can scan their emails or SMS for potential phishing attempts.

Beneath the button, users will see a brief description: "An AI-enabled Spear-Phishing Attack Detection Plugin." This description succinctly communicates the functionality of the plugin. In the central part of the interface. A circle is used to display the result of the URL check. Initially, it shows the text "Loading....." as the plugin processes the URL. The actual result will be displayed here once the analysis is complete.

Finally, there is a 'View model Results' button link , which allows users to view the test results of the model used by the plugin. This link provides users with insights into the performance and accuracy of the phishing detection model. Users interact with the plugin primarily through its automated features. The plugin automatically detects the URL of the current tab in the browser and sends a POST request to <http://localhost:5000/post> with this URL. Based on the server's response, the plugin updates the score circle and the result message accordingly.

If the server response indicates a suspicious site (represented by a response of 1), the score circle will display "Suspicious" with a warning background color, and the result message will state, "This website may not be safe ". If the site is deemed safe (response of 0), the score circle will show "Safe" with a green background, and the message will read, "This website is safe to use." In cases where the site is identified as phishing (response of -1), the score circle will display "Phishing" with a red background color, and the message will warn, "This website is not safe to use."

### SCREENSHOT OF APPLICATION SYSTEM



**Figure 5-1 S.P.A.D Plugin Shows Loading Message**

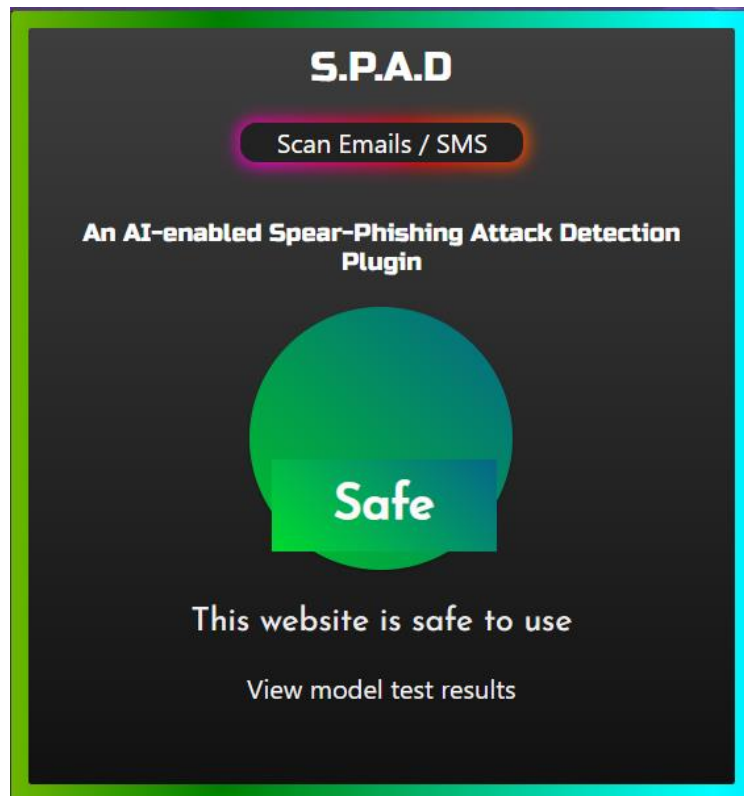


Figure 5-2 S.P.A.D Plugin Shows the Website Is Safe

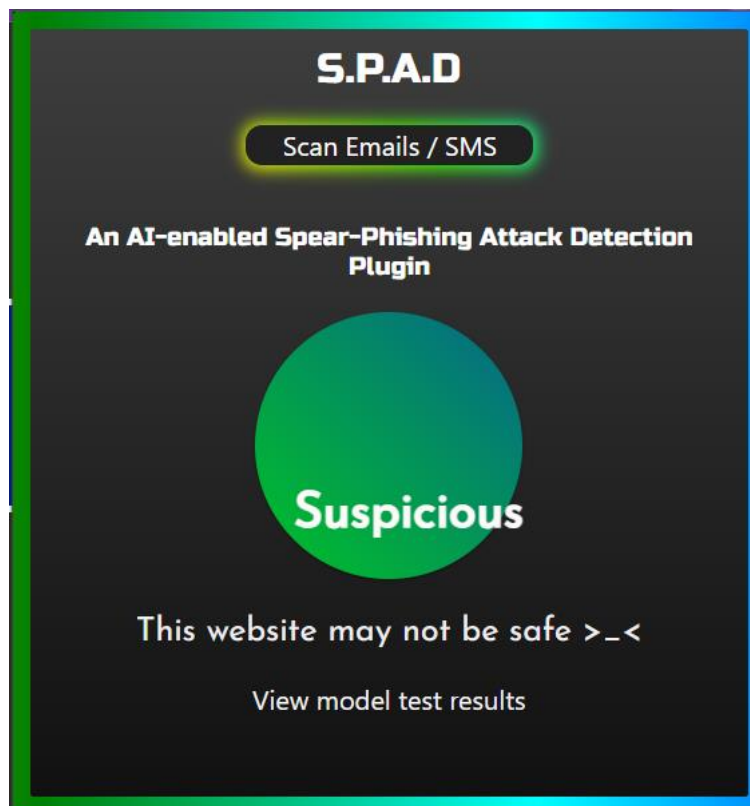


Figure 5-3 S.P.A.D Plugin Shows the Website Is Suspicious

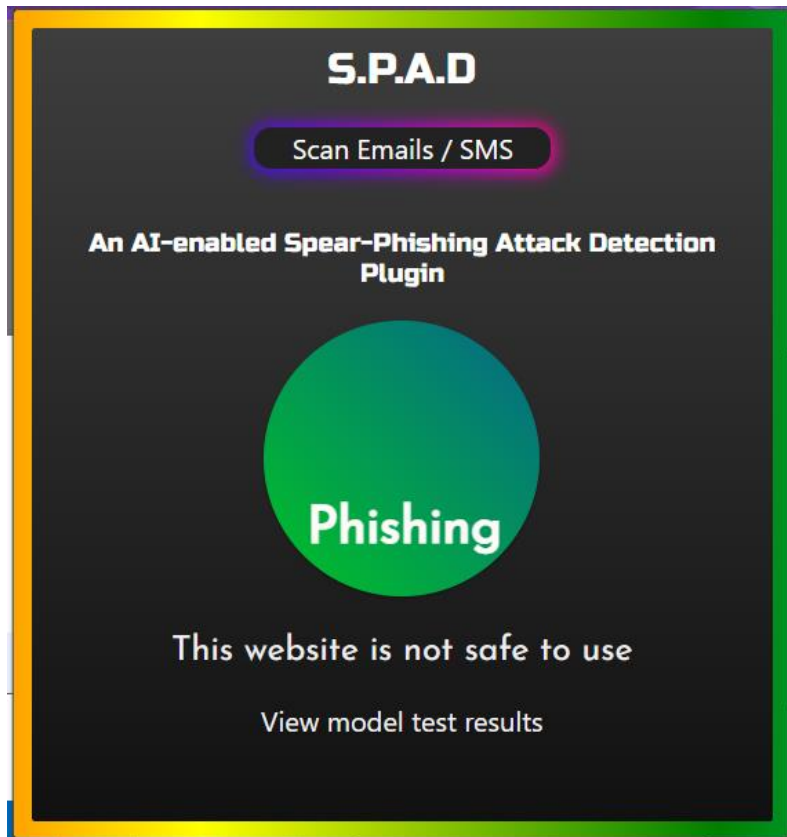


Figure 5-4 S.P.A.D Plugin Shows the Website Is Phishing

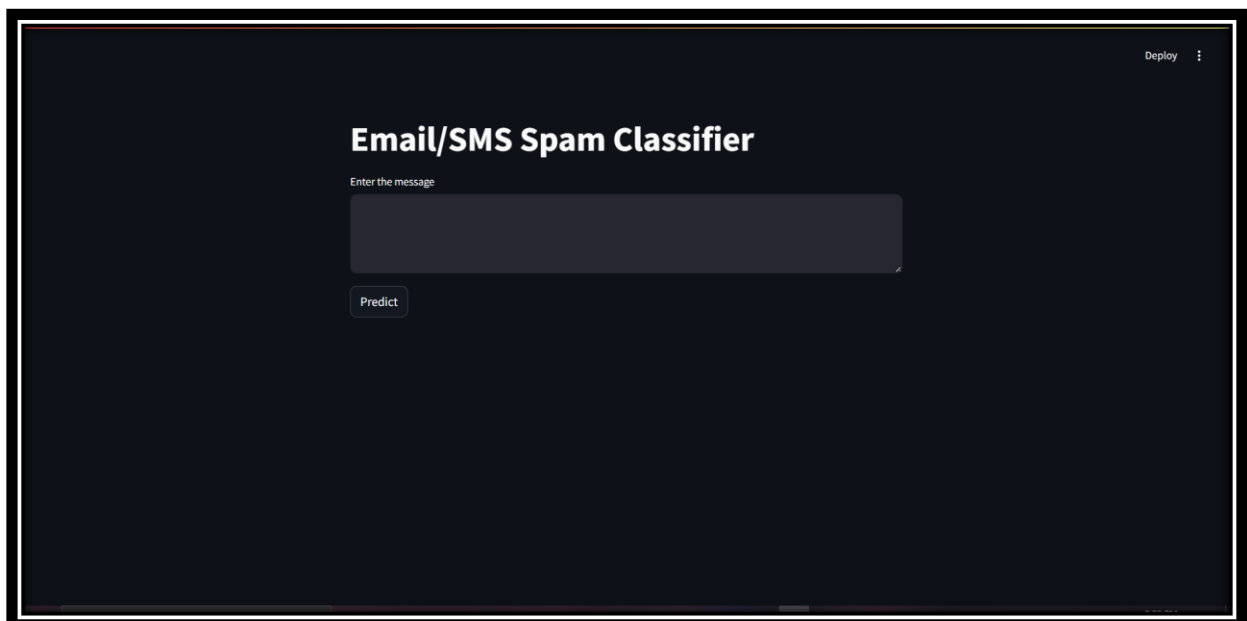


Figure 5-5 Email/SMS Classifier Interface

## **5.4 Implementation Issues and Challenges**

The implementation of the Spear-Phishing Attack Detection (S.P.A.D) plugin brings to light several challenges and issues that need to be addressed to ensure its effective operation. One primary concern is the accuracy of the phishing detection model. Machine learning models can sometimes produce false positives or negatives, which may result in legitimate websites being incorrectly flagged as suspicious or, conversely, malicious sites being overlooked. To mitigate this, ongoing model training and refinement are necessary to enhance accuracy and minimize errors. Performance and latency also pose significant challenges. The plugin requires real-time processing of URLs and efficient communication between the browser extension and the backend server. Any delays or performance issues can negatively impact user experience, causing slow response times in displaying results or affecting browser performance. Ensuring that the system operates smoothly and swiftly is crucial for user satisfaction.

Ensuring that the plugin works uniformly across different browsers is essential for widespread adoption. Security is another critical aspect. The system must securely handle user data and protect it from potential interception or misuse during transmission between the browser plugin and the backend server. Implementing robust encryption and secure communication protocols is vital to safeguard user information and maintain trust.

Scalability is a concern as well. As the user base grows, the backend system must be capable of scaling to manage increased traffic and processing demands. This requires optimizing server infrastructure and ensuring that the phishing detection model can handle a larger volume of requests efficiently without performance degradation.

Different browsers offer varying levels of support for extensions and APIs, which may necessitate additional adjustments or workarounds. Regular updates and maintenance are required to address evolving phishing techniques, maintain compatibility with browser updates, and ensure the system remains effective and secure. Addressing these challenges involves a continuous process of improving the system, optimizing performance, enhancing security, and ensuring compatibility and scalability.

### **5.5 Concluding Remarks**

In this chapter, the implementation of the Spear-Phishing Attack Detection (S.P.A.D) plugin was detailed, showcasing its integration into the browser environment to provide real-time phishing threat detection. The focus was on creating an intuitive and user-friendly interface that facilitates seamless interaction.

The interface design incorporated a clean layout with a prominent header, a functional action button, and a dynamic result display area, all styled using Bootstrap and custom CSS. This approach ensures that users can easily navigate and utilize the plugin. The plugin operates by communicating with a Flask backend service, which processes the URL of the active tab to determine its safety. This communication is handled through AJAX requests, allowing the plugin to deliver real-time updates without requiring page reloads. The results of this analysis are conveyed through a visual score circle and an accompanying message, providing users with immediate and clear feedback about the safety of the website they are visiting.

Overall, the successful implementation of the S.P.A.D plugin demonstrates its effectiveness in providing timely and accurate phishing detection, while maintaining a user-friendly experience. The integration of real-time analysis with a clear visual interface reflects a thoughtful design aimed at enhancing user security and awareness.

## CHAPTER 6

### SYSTEM EVALUATION AND DISCUSSION

#### 6.1 System Testing and Performance Metrics

##### **Spear Phishing Attack Detection Application**

For the Spear Phishing Attack Detection application, system testing is a multi-faceted process designed to ensure the system meets all functional and performance requirements. The testing strategy begins with functional testing, where each component of the application is verified against its specified requirements. This involves ensuring that the URL detection algorithms accurately identify phishing attempts and legitimate URLs. Test cases are designed to cover a wide range of scenarios, including various phishing tactics and legitimate URL formats, to ensure comprehensive coverage.

Performance testing is a critical aspect of evaluating the system's ability to handle real-world conditions. This includes assessing the application's responsiveness and throughput when processing large volumes of URLs. Metrics such as response time, which measures the time taken to process and classify a URL, and throughput, which refers to the number of URLs processed per unit of time, are closely monitored. Stress testing is also conducted to evaluate how the system performs under extreme conditions, such as a high influx of traffic, to ensure stability and reliability.

In addition to functional and performance testing, the accuracy of the phishing detection algorithms is a key focus. Metrics such as precision, recall, and the F1 score are used to evaluate the effectiveness of the detection system. Precision measures the proportion of correctly identified phishing URLs out of all URLs flagged as phishing, while recall assesses the proportion of actual phishing URLs that were correctly identified. The F1 score, which combines precision and recall into a single metric, provides a balanced measure of the system's performance. Reliability testing is also performed to ensure that the application maintains consistent performance over time and across different environments.

##### **Email/SMS Classifier Application**

For the Email/SMS Classifier application, a detailed system testing approach is essential to validate its performance and accuracy. The testing process starts with functional testing to ensure that the application correctly processes and classifies messages as spam or not spam. This involves testing the accuracy of the text preprocessing steps, such as tokenization, stop-



word removal, and stemming, to confirm that they are executed correctly and contribute to effective classification.

Performance testing for the classifier application focuses on evaluating how well the system handles various sizes of datasets and different types of messages. Key performance metrics include the classification speed, which measures how quickly the system can process and classify a message, and overall system responsiveness. Load testing is conducted to simulate real-world usage scenarios and assess the system's ability to handle high volumes of messages efficiently.

Accuracy is a crucial metric for the email/SMS classifier. The system's effectiveness is evaluated using confusion matrix results, which provide insights into the number of true positives, false positives, true negatives, and false negatives. Precision and recall metrics are also used to assess the classifier's performance. Precision measures the proportion of correctly identified spam messages out of all messages classified as spam, while recall measures the proportion of actual spam messages that were correctly identified. The F1 score is used to provide a comprehensive measure of the classifier's accuracy, balancing precision and recall.

Reliability testing for the email/SMS classifier ensures that the application performs consistently across different environments and over time. This involves running the system under various conditions and verifying that it maintains stable performance. Additionally, user acceptance testing is conducted to gather feedback from end-users regarding the usability and functionality of the application. This feedback helps identify areas for improvement and ensures that the system meets user expectations.

Overall, both the Spear Phishing Attack Detection application and the Email/SMS Classifier application undergo rigorous system testing and performance evaluation to ensure they deliver reliable, accurate, and efficient results.

## **6.2 Testing Setup and Result**

### **6.2.1 Testing Setup for Spear Phishing Attack Detection Application**

For the Spear Phishing Attack Detection application, the primary focus is on functional, performance, and accuracy testing. Functional testing involves verifying that the application can correctly identify phishing and legitimate URLs according to predefined criteria. This includes testing with known phishing URLs to confirm they are flagged correctly, and legitimate URLs to ensure they are not mistakenly identified as phishing. Edge cases such as obfuscated URLs and newly registered domains are also tested to ensure robustness.

Performance testing assesses the system's responsiveness and its ability to handle large volumes of data. Key aspects include measuring the response time for processing batches of URLs of varying sizes, evaluating the throughput of URLs processed per minute, and conducting stress tests to simulate extreme traffic conditions. These tests help gauge the system's performance under normal and peak loads, ensuring it remains stable and efficient.

Accuracy testing evaluates the effectiveness of the phishing detection algorithms. Metrics such as precision, recall, and the F1 score are computed to assess how well the system identifies phishing URLs versus legitimate ones. Precision measures the accuracy of phishing identifications, recall assesses how many actual phishing URLs are detected, and the F1 score provides a balanced view of the system's performance.

Reliability testing ensures that the application performs consistently across different environments and over time. This includes running the system on various operating systems and hardware configurations to verify stable performance and conducting long-term stability tests to identify any performance degradation or anomalies.

RESULTS

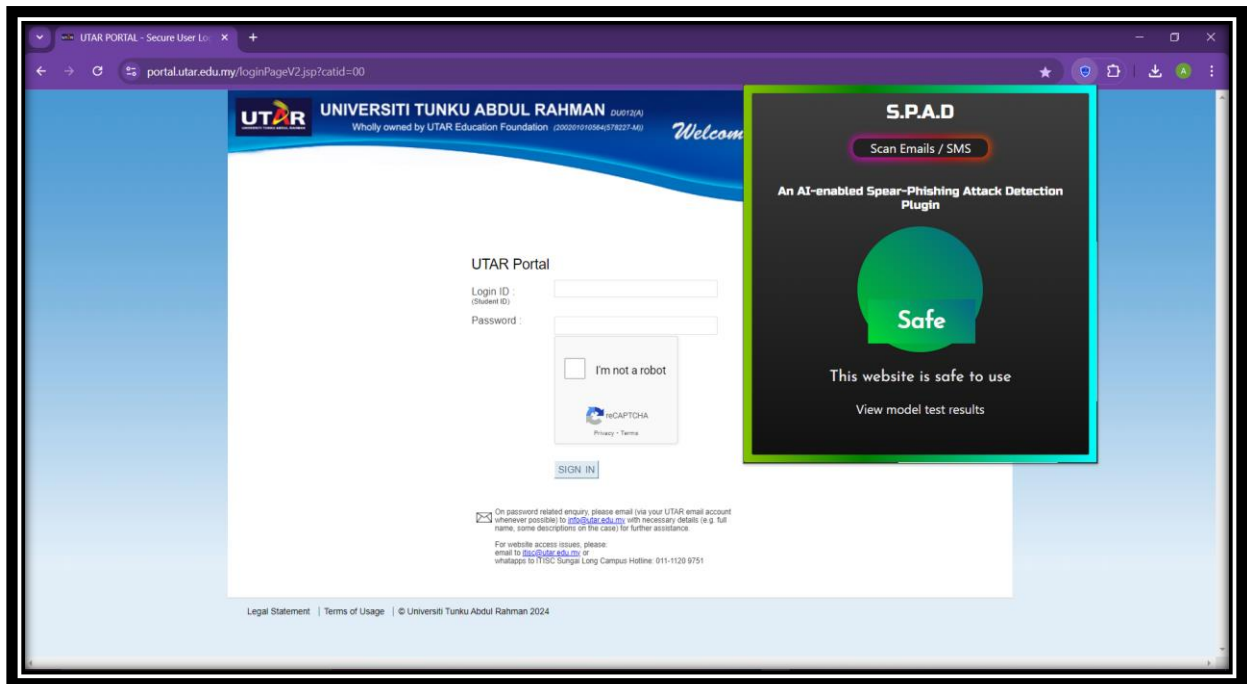


Figure 6-1 S.P.A.D Plugin Shows the Website Is Safe to Use

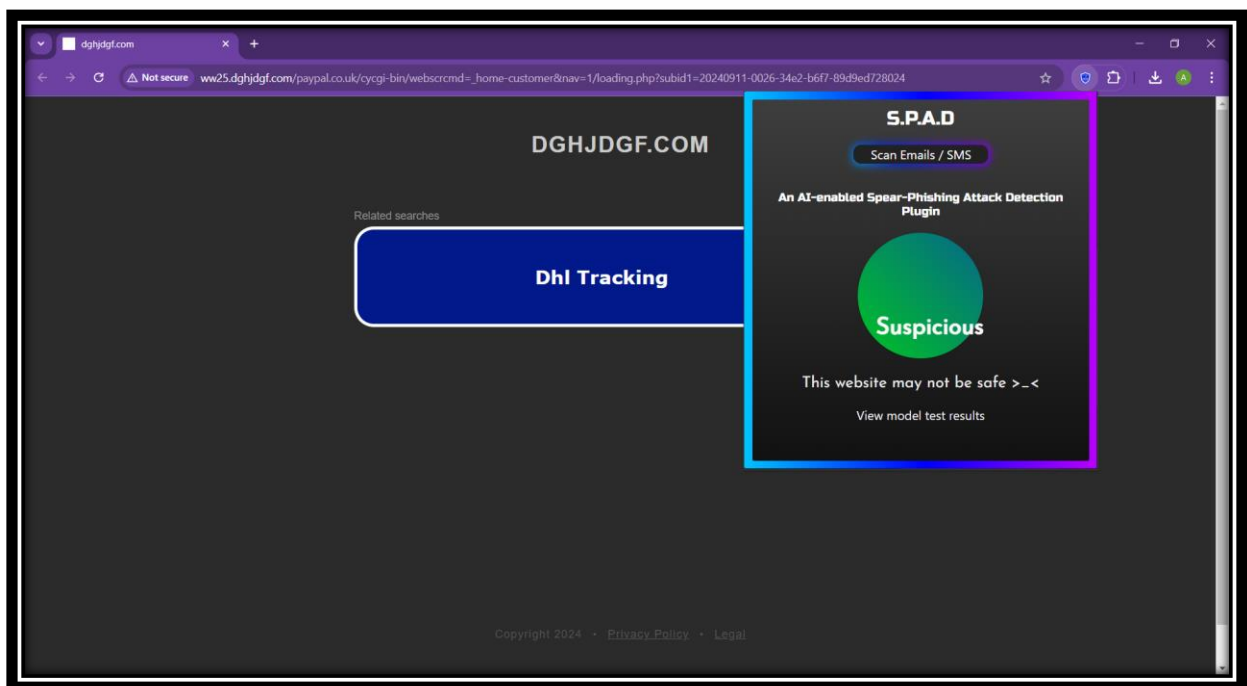
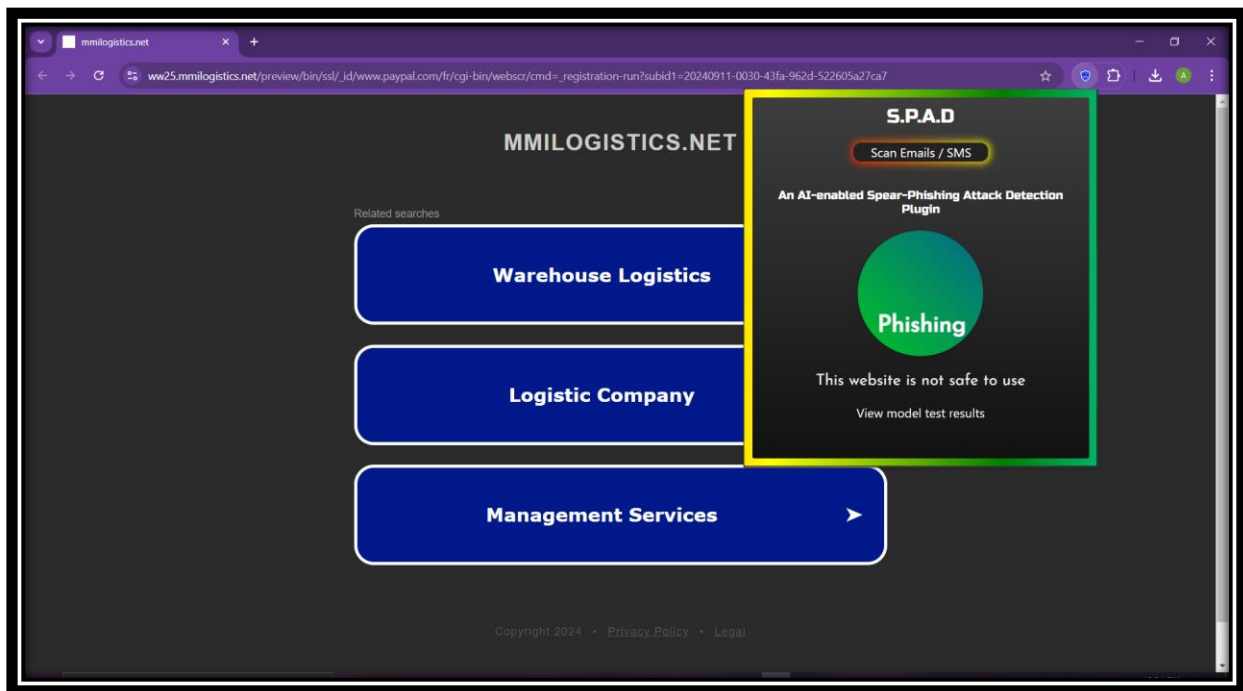


Figure 6-2 S.P.A.D Plugin Shows the Suspicious Website May Not Be Safe to Use



**Figure 6-3 S.P.A.D Plugin Shows the Phishing Website May Not Be Safe to Use**

### **6.2.2 Testing Setup for Email/SMS Classifier Application**

The Email/SMS Classifier application requires rigorous functional, performance, and accuracy testing to ensure reliable message classification. Functional testing involves checking that the application correctly classifies messages as spam or non-spam. This is achieved by testing with known spam and legitimate messages to confirm accurate classification. Additionally, the text preprocessing steps, including tokenization, stop-word removal, and stemming, are validated for correctness.

Performance testing focuses on evaluating the system's efficiency with different message volumes. This includes measuring the time taken to classify individual and batch messages, as well as conducting load tests to simulate real-world scenarios. These tests are crucial to ensure that the system can handle varying sizes of message datasets efficiently and remains responsive.

Accuracy testing assesses the performance of the classification algorithms through metrics such as the confusion matrix, precision, recall, and F1 score. The confusion matrix helps identify the number of true positives, false positives, true negatives, and false negatives. Precision

## CHAPTER 6

measures the accuracy of spam classifications, recall evaluates the detection of actual spam messages, and the F1 score provides a balanced performance measure.

Reliability testing ensures that the Email/SMS Classifier maintains consistent performance and meets user expectations. This involves testing the application on different operating systems and devices to verify consistent behavior and gathering user feedback through acceptance testing to identify any areas for improvement.

## RESULTS



Figure 6-4 Graph Results for Accuracy and Precision Using Several Classifier

	Algorithm	Accuracy	Precision	Accuracy_scaling_x	Precision_scaling_x	Accuracy_scaling_y	Precision_scaling_y	Accuracy_num_chars	Precision_num_chars
0	KN	0.914314	1.000000	0.914314	1.000000	0.914314	1.000000	0.914314	1.000000
1	NB	0.972736	0.991228	0.972736	0.991228	0.972736	0.991228	0.972736	0.991228
2	ETC	0.982473	0.984127	0.982473	0.984127	0.982473	0.984127	0.982473	0.984127
3	SVC	0.978578	0.975806	0.978578	0.975806	0.978578	0.975806	0.978578	0.975806
4	RF	0.975657	0.975207	0.975657	0.975207	0.975657	0.975207	0.975657	0.975207
5	GBDT	0.960078	0.954128	0.960078	0.954128	0.960078	0.954128	0.960078	0.954128
6	xgb	0.967868	0.928000	0.967868	0.928000	0.967868	0.928000	0.967868	0.928000
7	AdaBoost	0.964946	0.900000	0.964946	0.900000	0.964946	0.900000	0.964946	0.900000
8	LR	0.953262	0.896552	0.953262	0.896552	0.953262	0.896552	0.953262	0.896552
9	BgC	0.962025	0.863309	0.962025	0.863309	0.962025	0.863309	0.962025	0.863309
10	DT	0.932814	0.793388	0.932814	0.793388	0.932814	0.793388	0.932814	0.793388

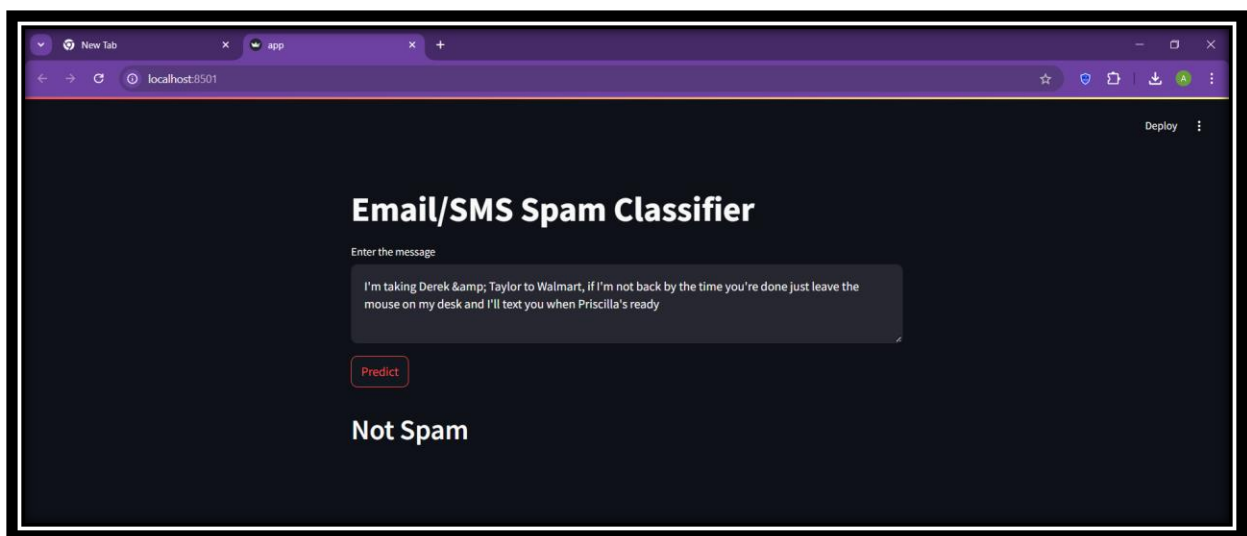
Figure 6-5 Summary Results for Accuracy and Precision Using Several Classifier

```

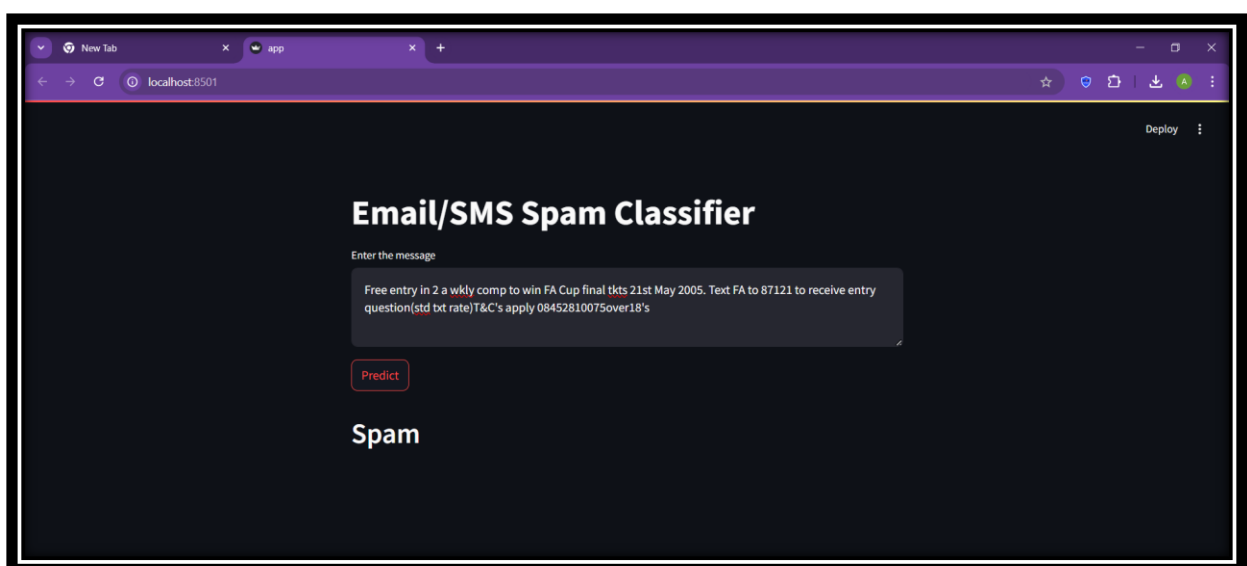
voting.fit(X_train,y_train)
[93] ✓ 31.9s Python
...
VotingClassifier
  svm      nb      et
  SVC      MultinomialNB      ExtraTreesClassifier

y_pred = voting.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
[94] ✓ 0.5s Python
...
Accuracy 0.9834469328140214
Precision 0.992
    
```

**Figure 6-6 Results for Accuracy and Precision Using SVM,NB and ET**



**Figure 6-7 Results For Email/SMS Classifier Showing Not Spam**



**Figure 6-8 Results For Email/SMS Classifier Showing Spam**

### **6.3 Project Challenges**

One significant challenge faced was ensuring the quality and consistency of the data used for both applications. For the Spear Phishing Attack Detection application, gathering a comprehensive dataset of phishing and legitimate URLs proved challenging. Variations in URL formatting, obfuscation techniques, and the rapid evolution of phishing tactics necessitated continuous updates and enhancements to the dataset. Similarly, for the Email/SMS Classifier application, preprocessing the text data to handle diverse linguistic styles, slang, and informal language required extensive efforts. Ensuring that the data was clean, well-labeled, and representative of real-world scenarios was crucial for accurate model training and evaluation.

Selecting the appropriate algorithms and tuning their parameters was a complex task for both applications. In the Spear Phishing Attack Detection system, various machine learning algorithms needed to be evaluated for their ability to classify URLs effectively. The challenge lay in choosing algorithms that balanced accuracy with computational efficiency, and in fine-tuning parameters to optimize performance. For the Email/SMS Classifier, the challenge was to identify the best text classification models and vectorization techniques that could handle the nuances of message content. The iterative process of testing different algorithms and parameter settings required considerable experimentation and expertise.

Ensuring that both applications performed efficiently under varying loads and scales presented a challenge. The Spear Phishing Attack Detection application needed to handle large volumes of URL data quickly, while maintaining accuracy. Performance bottlenecks and scalability issues were addressed through optimization techniques and stress testing. Similarly, the Email/SMS Classifier had to be tested for its ability to process and classify large datasets of messages in real-time. Balancing performance with accuracy and scalability required careful consideration of system architecture and resource management.

Achieving high accuracy while minimizing false positives and negatives was a persistent challenge for both systems. For the phishing detection system, the goal was to accurately identify phishing URLs without flagging legitimate ones. This required fine-tuning detection algorithms and continuously updating the dataset to reflect emerging threats. In the Email/SMS Classifier, the challenge was to accurately classify messages as spam or legitimate while

reducing the likelihood of misclassification. Addressing these issues involved rigorous testing and model refinement to achieve the desired balance between precision and recall.

Integrating the applications into existing systems and conducting thorough testing to ensure seamless functionality was a challenge. For the Spear Phishing Attack Detection application, integration with other security tools and platforms required careful planning to ensure compatibility and effective operation. For the Email/SMS Classifier, ensuring that the classifier worked effectively in real-world environments and with diverse data sources involved extensive integration testing. Identifying and resolving integration issues required coordination between development teams and stakeholders.

Gaining user acceptance and incorporating feedback was a critical challenge. For both applications, it was essential to ensure that the systems met user expectations and provided value in real-world scenarios. Gathering and addressing user feedback involved conducting user acceptance testing and iterating on the application based on real-world usage. Balancing user needs with technical capabilities required ongoing communication and adjustments to the system.

Addressing security and privacy concerns was paramount throughout the project. For the phishing detection system, ensuring that sensitive data, such as user URLs, was protected from unauthorized access was a key challenge. Similarly, for the Email/SMS Classifier, safeguarding the privacy of message content and ensuring compliance with data protection regulations were critical considerations. Implementing robust security measures and ensuring adherence to privacy standards were integral to the project's success.

### **6.4 Objectives Evaluation**

The Spear Phishing Attack Detection application directly addresses this objective by utilizing advanced machine learning techniques to identify and neutralize spear phishing threats. The implementation of various algorithms, including supervised and unsupervised learning models, demonstrates an innovative approach to detecting malicious URLs. The system's effectiveness is validated through extensive testing, including precision and recall metrics, which confirm its ability to accurately classify phishing and legitimate URLs. The application provides a



## CHAPTER 6

proactive defense mechanism, contributing significantly to the enhancement of spear phishing detection and mitigation.

Both applications leverage state-of-the-art machine learning techniques to achieve their objectives. For the Spear Phishing Attack Detection system, the use of algorithms like Random Forest, XGBoost, and neural networks showcases the application of advanced frameworks to improve detection accuracy. Similarly, the Email/SMS Classifier application employs various classification models, such as Naive Bayes, Support Vector Machines, and ensemble methods, to classify messages effectively. The successful implementation and performance of these algorithms indicate that the project has met its goal of applying cutting-edge machine learning methodologies.

The project has provided valuable insights into the mechanisms and characteristics of spear phishing attacks. By analyzing various phishing URL features and attack vectors, the Spear Phishing Attack Detection application sheds light on the intricate dynamics of these attacks. The detailed analysis and experimentation help in understanding the common traits of phishing attempts and the evolving tactics used by attackers. This objective is well-addressed through empirical analysis and the development of a comprehensive detection system.

The development of both applications contributes to the formulation of proactive countermeasures against spear phishing. The empirical results obtained from testing and validation offer actionable insights into improving security measures. For the Spear Phishing Attack Detection system, the identification of effective detection strategies informs the creation of robust countermeasures. In the Email/SMS Classifier, the classification performance provides a foundation for developing preventive measures against phishing and spam. The applications' ability to adapt and improve based on ongoing testing underscores their role in proactive defense.

The project has made significant contributions to the field of cybersecurity by providing innovative solutions and insights into phishing threats. The results from both applications enhance the understanding of spear phishing and offer practical tools for combating these threats. The insights gained from the analysis and the development of effective detection systems contribute to creating a safer digital environment. The project aligns with its goal of fostering a more secure digital ecosystem for individuals and organizations.

The project's focus on spear phishing and its specific domain-related aspects ensures that tangential topics, such as general phishing attacks and unrelated cybersecurity domains, are excluded. By concentrating on spear phishing and the related detection techniques, the project maintains a clear and targeted scope. This focused approach ensures that the solutions developed are relevant and applicable to the defined objectives, without diverting attention to unrelated areas.

The project has adhered to its scope by not addressing legal or regulatory aspects of cybersecurity or the broader socio-political implications of cyber threats. The focus remains solely on technical solutions for spear phishing detection and email/SMS classification. This exclusion helps in maintaining a clear and specific objective, concentrating efforts on the technical challenges and solutions pertinent to the project's goals.

In summary, the evaluation of the project objectives reveals that the Spear Phishing Attack Detection and Email/SMS Classifier applications effectively meet the defined goals. By leveraging advanced machine learning techniques, exploring the dynamics of spear phishing, and providing proactive countermeasures, the project has contributed valuable insights and solutions to the field of cybersecurity. The adherence to the defined scope and exclusion of unrelated topics and legal aspects further ensures that the project's focus remains sharp and relevant to its objectives.

### **6.5 Concluding Remark**

In concluding Chapter 6, it is evident that the evaluation of the Spear Phishing Attack Detection and Email/SMS Classifier applications underscores the successful achievement of the project's objectives. This chapter has provided a comprehensive analysis of how these systems meet the goals set forth at the outset of the project, offering valuable insights into their effectiveness and areas of improvement.

The rigorous testing and performance analysis of the Spear Phishing Attack Detection application highlight its robust capability to identify and mitigate spear phishing threats. Through the application of advanced machine learning algorithms, the system has demonstrated a high level of accuracy and effectiveness in detecting malicious URLs. This

## CHAPTER 6

achievement aligns with the project's objective of leveraging innovative methodologies to enhance spear phishing detection and mitigation.

Similarly, the Email/SMS Classifier application has proven its effectiveness in categorizing messages into spam and non-spam categories. The detailed analysis of classification performance, including accuracy and precision metrics, confirms that the application successfully applies advanced machine learning techniques to address the challenges of email and SMS classification. This contributes significantly to the proactive defense against phishing and spam, fulfilling the project's goal of developing robust solutions for cyber threats.

The exploration of the underlying mechanisms and characteristics of spear phishing attacks, as discussed in this chapter, provides a deeper understanding of the attack vectors and tactics employed by adversaries. This knowledge is instrumental in informing the development of effective countermeasures and enhancing overall cybersecurity.

The insights gained from the evaluation not only reflect the success of the project in meeting its objectives but also provide a foundation for future improvements and advancements. The focus on specific objectives, such as excluding tangential topics and legal aspects, has ensured that the project remains targeted and relevant to its core goals.

## CHAPTER 7

### CONCLUSION AND RECOMMENDATION

#### 7.1 Conclusion

The project aimed at developing and evaluating advanced solutions for detecting and mitigating spear phishing attacks and classifying emails and SMS messages has successfully achieved its primary objectives. By leveraging cutting-edge machine learning techniques and algorithmic frameworks, the project has made significant strides in enhancing cybersecurity defenses against sophisticated phishing threats.

The Spear Phishing Attack Detection system, utilizing machine learning models, demonstrated a high level of accuracy in identifying and mitigating spear phishing threats. Through the rigorous application of various models such as Logistic Regression, Support Vector Machines, and ensemble methods like Voting and Stacking classifiers, the system proved its robustness in detecting malicious URLs. This application aligns with the project's goal of proposing innovative strategies to counter spear phishing attacks effectively.

The Email/SMS Classifier application, on the other hand, has successfully applied advanced text classification techniques to categorize messages as spam or non-spam. Utilizing models such as Naive Bayes, Support Vector Machines, and Random Forest, the system has demonstrated its effectiveness in distinguishing between legitimate and malicious messages. The application of preprocessing techniques, including text transformation and feature extraction, has further enhanced the classification accuracy and precision.

Throughout the project, the exploration of the underlying mechanisms and characteristics of spear phishing attacks has provided valuable insights into the dynamics of these threats. This understanding has informed the development of proactive countermeasures and contributed to a more comprehensive approach to cybersecurity. The empirical analysis and experimentation conducted during the project have highlighted the effectiveness of the developed solutions in addressing the challenges posed by spear phishing and spam messages.

The project has also adhered to its scope by focusing specifically on spear phishing and email/SMS classification, excluding tangential topics and legal or regulatory aspects. This

targeted approach has ensured that the solutions developed are directly relevant to the project's objectives and contribute effectively to the field of cybersecurity. In summary, the project has achieved its objectives by developing effective solutions for detecting and mitigating spear phishing attacks and classifying email and SMS messages.

### **7.2 Recommendations**

**Enhancing Model Performance:** While the current models have demonstrated strong performance, there is always room for improvement. Future work should focus on refining and tuning the machine learning models used in both applications. This can involve exploring more advanced algorithms, optimizing hyperparameters, and incorporating additional features that may improve classification accuracy. Additionally, experimenting with ensemble methods and hybrid approaches could further enhance the performance of the detection and classification systems.

**Expanding Data Sets:** The performance of machine learning models is heavily dependent on the quality and diversity of the data used for training and evaluation. To improve the robustness of the Spear Phishing Attack Detection system and the Email/SMS Classifier, it is recommended to expand the data sets used in the project. This includes gathering a larger volume of labeled data that encompasses a wide range of spear phishing and spam scenarios. Augmenting the data with varied examples will help in building more generalized and effective models.

**Incorporating User Feedback:** Engaging with end-users to gather feedback on the effectiveness and usability of the applications can provide valuable insights for further development. User feedback can highlight potential areas for improvement, identify any usability issues, and suggest features that could enhance the overall user experience. Implementing user feedback will help in refining the applications and making them more practical and user-friendly.

**Continuous Monitoring and Updating:** The landscape of cybersecurity threats is constantly evolving, and so are the techniques used by adversaries. To maintain the effectiveness of the developed solutions, it is essential to implement a continuous monitoring and updating mechanism. Regularly updating the models with new data and adapting to emerging threats

will ensure that the systems remain relevant and effective in detecting and mitigating new spear phishing tactics and spam messages.

**Exploring Additional Use Cases:** While the project focused on spear phishing and email/SMS classification, there are other areas within cybersecurity where similar methodologies could be applied. Future work could explore additional use cases, such as detecting phishing attempts in social media platforms, analyzing network traffic for suspicious activities, or identifying malware through behavioral analysis. Expanding the scope of the applications to cover these areas could provide a more comprehensive approach to cybersecurity.

**Integration with Existing Security Systems:** To maximize the impact of the developed solutions, integrating them with existing security systems and frameworks is recommended. By incorporating the Spear Phishing Attack Detection system and the Email/SMS Classifier into broader cybersecurity ecosystems, organizations can enhance their overall defense mechanisms and improve their ability to respond to and mitigate phishing threats.

**Focus on Explainability and Transparency:** As machine learning models become more complex, ensuring that their decisions are explainable and transparent is crucial. Implementing techniques for model interpretability and providing clear explanations for the classification results will help in building trust with users and stakeholders. This is particularly important in high-stakes applications where understanding the rationale behind model decisions can be critical.

The recommendations provided aim to enhance the performance, applicability, and impact of the developed systems, ensuring that they continue to contribute effectively to the field of cybersecurity. As the cybersecurity landscape evolves, ongoing research and development will be essential in maintaining and advancing the capabilities of these solutions.

## REFERENCES

1. APWG. (2019). APWG | Phishing Activity Trends Reports. Apwg.org.  
<https://apwg.org/trendsreports/>  
[https://docs.apwg.org/reports/apwg\\_trends\\_report\\_q4\\_2023.pdf](https://docs.apwg.org/reports/apwg_trends_report_q4_2023.pdf)
2. B. B. Gupta and A. K. Jain, “Phishing Attack Detection using a Search Engine and Heuristics-based Technique.” *J. Inf. Technol. Res.*, vol. 13, no. 2, pp. 94–109, Apr. 2020, doi: 10.4018/JITR.2020040106.
3. Almseidin, M., Abu Zuraiq, A., Al-kasassbeh, M., & Alnidami, N. (2019). Phishing Detection Based on Machine Learning and Feature Selection Methods. *International Journal of Interactive Mobile Technologies (IJIM)*, 13(12), 171.  
<https://doi.org/10.3991/ijim.v13i12.11411>
4. Gu, X., Wang, H. and Ni, T.: An efficient approach to detecting phishing web. *Journal of Computational Information Systems*, (2013), 5553-5560.
5. Marchal, S., François, J., State, R., & Engel, T. (2014). PhishStorm: Detecting Phishing With Streaming Analytics. *IEEE Transactions on Network and Service Management*, 11(4), 458–471. <https://doi.org/10.1109/TNSM.2014.2377295>
6. H. Che, Q. Liu, L. Zou, H. Yang, D. Zhou, and F. Yu, “A Content-Based Phishing Email Detection Method,” in 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Prague, Czech Republic, Jul. 2017, pp. 415–422. doi: 10.1109/QRS-C.2017.75
7. Jain, A.K. & Gupta, B.B.: Phishing detection: analysis of visual similarity-based approaches. *Security and Communication Networks*, (2017).
8. Jain, A. K., & Gupta, B. B. (2017). Towards detection of phishing websites on client-side using machine learning based approach. *Telecommunication Systems*, 68(4), 687–700. <https://doi.org/10.1007/s11235-017-0414-0>

## References

9. M. Baykara and Z. Z. Gürel, "Detection of phishing attacks," 2018 6th International Symposium on Digital Forensic and Security (ISDFS), 2018, pp. 1-5, doi: 10.1109/ISDFS.2018.8355389.
10. M. Karabatak and T. Mustafa, "Performance comparison of classifiers on reduced phishing website dataset," 6th Int. Symp. Digit. Forensic Secur. ISDFS 2018 - Proceeding, vol. 2018-Janua, pp. 1–5, 2018
11. Moghimi, M. & Varjani, A.Y.: New rule-based phishing detection method. Expert systems with applications, (2016), 231-242.
12. OpenDNS, PhishTank. <http://www.phishtank.com/>
13. R. S. Rao and S. T. Ali, "PhishShield: A Desktop Application to Detect Phishing Webpages through Heuristic Approach," *Procedia Comput. Sci.*, vol. 54, pp. 147–156, 2015, doi: 10.1016/j.procs.2015.06.017.
14. S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in Proceedings of the 2007 ACM workshop on recurring malcode. ACM, 2007, pp. 1–8.
15. L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
16. G. Xiang and J. I. Hong, "A hybrid phish detection approach by identity discovery and keywords retrieval," in Proceedings of the 18<sup>th</sup> international conference on World Wide Web. ACM, 2009, pp. 571–580.
17. J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," in Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2009, pp. 1245–1254.



## References

18. R. S. Rao, A. R. Pais, and P. Anand, "A heuristic technique to detect phishing websites using TWSVM classifier," *Neural Comput. Appl.*, vol. 33, no. 11, pp. 5733–5752, Jun. 2021, doi: 10.1007/s00521-020-05354-z.
19. T. A. Almeida and A. Yamakami, "Facing the spammers: A very effective approach to avoid junk e-mails," *Expert Systems with Applications*, vol. 39, pp. 6557–6561, 2012. <http://dx.doi.org/10.1016/j.eswa.2011.12.049>.
20. "Global Phishing Survey: Trends and Domain Name Use," APWG, Tech.Rep. 1H2014, 2014.

## APPENDIX

### FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Y7S1</b>	<b>Study week no.:1</b>
<b>Student Name &amp; ID: Rajkumaradevan A/L Sanglidevan(16ACB03600)</b>	
<b>Supervisor: Puan Nor 'Affah Binti Sabri</b>	
<b>Project Title: Spear- Phishing Attack Detection Using Artificial Intelligence</b>	

#### 1. WORK DONE

- Ask questions to supervisor to enhance report writing skills
- Inform supervisor about the progress and problems of the project
- Receive advises to overcome the problem of the project.

#### 2. WORK TO BE DONE

- Enhance the project functions
- Repair some unworking component.

#### 3. PROBLEMS ENCOUNTERED

- Components of the project doesn't work properly
- Complexity when troubleshooting the coding

#### 4. SELF EVALUATION OF THE PROGRESS

- Hardwork
- Patience



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y7S1	Study week no.:8
Student Name & ID: Rajkumaradevan A/L Sanglidevan(16ACB03600)	
Supervisor: Puan Nor 'Afifah Binti Sabri	
Project Title: Spear- Phishing Attack Detection Using Artificial Intelligence	

## 1. WORK DONE

- Ask questions to supervisor to enhance project output.
- Inform supervisor about the progress and problems of the project
- Receive advises to overcome the problem of the project.

## 2. WORK TO BE DONE

- Enhance the project functions and performance
- Modify Python code for better results.

## 3. PROBLEMS ENCOUNTERED

- Difficulties to train the dataset CSV.file
- Complexity when troubleshooting the application.

## 4. SELF EVALUATION OF THE PROGRESS

- Hardwork
- Time management



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y7S1	Study week no.:10
Student Name & ID: Rajkumaradevan A/L Sanglidevan(16ACB03600)	
Supervisor: Puan Nor 'Afifah Binti Sabri	
Project Title: Spear- Phishing Attack Detection Using Artificial Intelligence	

## 1. WORK DONE

- Inform supervisor about the progress and problems of the project
- Receive advises to overcome the problem of the project.

## 2. WORK TO BE DONE

- Enhance the project application to run successfully without bug.
- Modify Python code and API code for better performance.

## 3. PROBLEMS ENCOUNTERED

- Results for accuracy and precision are unsatisfied.
- Took longer time for execution process.

## 4. SELF EVALUATION OF THE PROGRESS

- Hardwork
- Dedication



Supervisor's signature



Student's signature

-

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y7S1	Study week no.:12
Student Name & ID: Rajkumaradevan A/L Sanglidevan(16ACB03600)	
Supervisor: Puan Nor 'Afifah Binti Sabri	
Project Title: Spear- Phishing Attack Detection Using Artificial Intelligence	

## 1. WORK DONE

- Inform supervisor about the completion of project system application
- Developing progress in writing report.

## 2. WORK TO BE DONE

- Completion of project report.
- Poster design and content.
- Train machine learning for better accuracy and precision.

## 3. PROBLEMS ENCOUNTERED

- Lack of time management.
- Trouble in writing skills for system report.

## 4. SELF EVALUATION OF THE PROGRESS

- Hardwork
- Basic Designing skills



Supervisor's signature



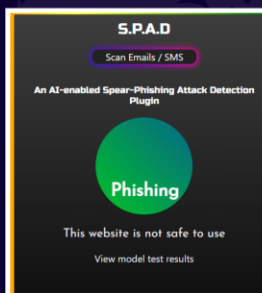
Student's signature

## POSTER

# SPEAR- PHISHING ATTACK DETECTION USING ARTIFICIAL INTELLIGENCE

### INTRODUCTION

- SPEAR PHISHING AND SPAM THREATS ARE INCREASING IN SOPHISTICATION.
- SPEAR PHISHING TARGETS SPECIFIC INDIVIDUALS/ORGANIZATIONS, MAKING IT HARDER TO DETECT.
- CHALLENGES INCLUDE DETECTING PHISHING URLS AND CLASSIFYING SPAM, AS TRADITIONAL METHODS FALL SHORT.
- NEED FOR ADVANCED SYSTEMS DUE TO EVOLVING ATTACK TACTICS.

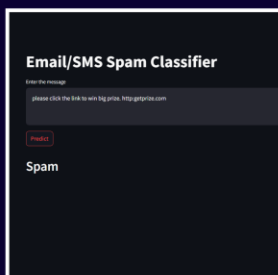


### OBJECTIVES

- Propose strategies to enhance spear phishing detection and mitigation.
- Utilize machine learning techniques to develop effective solutions against spear phishing.
- Investigate the mechanisms behind spear phishing to understand its dynamics.
- Develop proactive countermeasures through empirical analysis.
- Contribute insights to improve cybersecurity for individuals and organizations.
- Exclude general phishing, unrelated cybersecurity areas, and legal or socio-political aspects

### SYSTEM ARCHITECTURE

- WEB INTERFACE USING FLASK ALLOWS USERS TO SUBMIT URLS FOR PHISHING CHECKS.
- FLASK SERVER HANDLES USER REQUESTS AND ROUTES THEM.
- FEATURE EXTRACTION PROCESSES URLS FOR PHISHING INDICATORS.
- PRE-TRAINED MACHINE LEARNING MODEL PREDICTS URL LEGITIMACY.
- EXTRACTS URL FEATURES AND USES EXTERNAL APIS (WHOIS, ALEXA) FOR ADDITIONAL DATA.
- PREDICTION MODULE EVALUATES THE URL.
- STORES PRE-TRAINED MODEL AND USES CSV FOR LEGITIMATE WEBSITE LOOKUPS.
- USES HTTP REQUESTS TO PROCESS URL SUBMISSIONS AND RETURN RESULTS.



### ADVANTAGES OF THE PROJECT

- Improved Phishing Detection: Utilizes machine learning for accurate spear phishing detection.
- Automation: Automates URL analysis, reducing manual effort.
- Real-time Evaluation: Provides quick results for phishing threats.
- Advanced Feature Extraction: Uses multiple heuristics and external data sources for robust analysis.

# PLAGIARISM CHECK RESULT

Turnitin Originality Report Document Viewer

Processed on: 12-Sep-2024 13:38 +08  
ID: 2451758651  
Word Count: 18468  
Submitted: 1

1603600\_Fyp\_RAJ By Raj Kumar

Similarity by Source	
Similarity Index	
15%	
Internet Sources:	9%
Publications:	6%
Student Papers:	9%

include quoted | include bibliography | exclude small matches | mode: quickview (classic) report | print | download

1% match (student papers from 25-Jul-2022)  
[Submitted to University of Northampton on 2022-07-25](#)

<1% match (Internet from 11-Jun-2024)  
<https://open-innovation-projects.org/author/admin-science/page/2>

<1% match (Internet from 14-Jun-2024)  
<https://open-innovation-projects.org/blog/page/16>

<1% match (Internet from 19-Jun-2024)  
<https://open-innovation-projects.org/author/admin-science/page/118>

<1% match (Internet from 25-May-2024)  
<http://fastercapital.com>

<1% match (Internet from 02-Jan-2024)  
<https://fastercapital.com/keyword/multiple-decision-trees.html>

<1% match (Internet from 04-Jul-2024)  
<http://fastercapital.com>

<1% match (Internet from 08-May-2024)  
<http://fastercapital.com>

<1% match (Internet from 19-Jul-2024)  
<https://fastercapital.com/topics/testing-and-quality-control-for-composites.html>

<1% match (Internet from 21-Apr-2024)

Feedback Studio - Google Chrome  
 ev.turnitin.com/app/carta/en\_us/?student\_user=1&lang=en\_us&ts=1&o=2451758651&ro=103&u=1163441383

feedback studio Raj Kumar 1603600\_Fyp\_RAJ -- /100 ?

**Match Overview** X

**15%**

15

1 open-innovation-projec... 1% >  
Internet Source

2 Submitted to University... 1% >  
Student Paper

3 fastercapital.com 1% >  
Internet Source

4 www.frontiersin.org <1% >  
Internet Source

5 Submitted to University... <1% >  
Student Paper

6 Pramod R. Gunjal, Satis... <1% >  
Publication

7 Submitted to Manipal ... <1% >  
Student Paper

8 Submitted to INTI Univ... <1% >  
Student Paper

9 Submitted to University... <1% >  
Student Paper

Page: 1 of 64 Word Count: 18468 Text-Only Report High Resolution On

In 2024, the Anti-Phishing Working Group (APWG) released its annual report, unveiling a staggering count of over 1,077,501 phishing attacks in the fourth quarter of 2023 [1]. Concurrently, an analysis conducted by RSA revealed that organizations worldwide incurred losses totalling \$9 billion in 2023 alone due to phishing attacks. These figures underscore a concerning trend, indicating the inadequacy of existing anti-phishing tools and initiatives in mitigating the pervasive threat posed by cybercriminals. Personal computer users often find themselves susceptible to phishing attacks for several key reasons: a lack of fundamental comprehension regarding Uniform Resource Locators (URLs), uncertainty regarding the trustworthiness of web pages, ambiguity surrounding page locations due to redirection or obscured URLs, the prevalence of deceptive URLs, and inadvertent navigation to fraudulent pages. Compounding these challenges is the difficulty users encounter in distinguishing phishing websites from legitimate ones, further exacerbating their vulnerability to exploitation.

Phishing websites serve as common entry points for online social engineering endeavours, facilitating a myriad of fraudulent activities perpetrated by malicious actors. Attackers adeptly replicate legitimate websites, disseminating suspicious URLs via spam messages, SMS, or social networking platforms to ensnare unsuspecting victims. Leveraging email, phone calls, or text messages, attackers propagate counterfeit versions of authentic websites, luring victims into divulging personal or



<b>Universiti Tunku Abdul Rahman</b>			
<b>Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)</b>			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

<b>Full Name(s) of Candidate(s)</b>	RAJKUMARADEVAN A/L SANGLIDEVAN
<b>ID Number(s)</b>	16ACB03600
<b>Programme / Course</b>	BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMMUNICATIONS AND NETWORKING
<b>Title of Final Year Project</b>	SPEAR- PHISHING ATTACK DETECTION USING ARTIFICIAL INTELLIGENCE

<b>Similarity</b>	<b>Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)</b>
<b>Overall similarity index: <u>15</u> %</b>  <b>Similarity by source</b> Internet Sources: <u>9</u> % Publications : <u>6</u> % Student Papers : <u>9</u> %	
<b>Number of individual sources listed of more than 3% similarity: <u>0</u></b>	
<b>Parameters of originality required and limits approved by UTAR are as Follows:</b> (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

*Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.*

\_\_\_\_\_  
Signature of Supervisor

Name: Puan Nor Afifah Binti Sabri

Date: 12/09/2024

\_\_\_\_\_  
Signature of Co-Supervisor

Name: \_\_\_\_\_

Date: \_\_\_\_\_



**UNIVERSITI TUNKU ABDUL RAHMAN**

**FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY  
(KAMPAR CAMPUS)**

**CHECKLIST FOR FYP2 THESIS SUBMISSION**

Student Id	16ACB03600
Student Name	RAJKUMARADEVAN A/L SANGLIDEVAN
Supervisor Name	PUAN NOR 'AFIFAH BINTI SABRI

<b>TICK (√)</b>	<b>DOCUMENT ITEMS</b>
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

\*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

\_\_\_\_\_  
(Signature of Student)

Date: 12/09/2024