



ALBUKHARY INTERNATIONAL UNIVERSITY

ALBUKHARY INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTING AND INFORMATICS

COURSE DETAILS	
SCHOOL	SCHOOL OF COMPUTING AND INFORMATICS
PROJECT 1 COORDINATOR	NADIAH ARSAT
COURSE NAME	PROJECT 2 - FINAL REPORT
COURSE CODE	CCC 3026
CLO	CLO 1: Formulate computer science project to solve problems in the community and industry (C6, PLO2).
WEIGHTAGE	50%
STUDENT NAME	Md Saiful Arefin, Mutasim Billah, Abdullah Al Hadi
ID	AIU21102219, AIU21102122, AIU21102089
SEMESTER	2 2025/2026
PROJECT TITLE	Phishing Attack Detection Using NLP and Deep Learning
SUPERVISOR	Assoc. Prof. Dr. Basheer Riskhan

# **Phishing Attack Detection Using NLP and Deep Learning**

**MD SAIFUL AREFIN, MUTASIM BILLAH,  
ABDULLAH AL HADI**

**BACHELOR OF COMPUTER SCIENCE  
(HONOURS)  
ALBUKHARY INTERNATIONAL UNIVERSITY  
2025**

## Table of Contents

<b>DECLARATION</b>	<b>i</b>
<b>APPROVAL</b>	<b>ii</b>
<b>ACKNOWLEDGMENT</b>	<b>iii</b>
<b>TABLE OF CONTENT</b>	<b>2</b>
<b>LIST OF TABLES</b>	<b>2</b>
<b>LIST OF FIGURES</b>	<b>4</b>
<b>LIST OF ABBREVIATIONS</b>	<b>5</b>
<b>ABSTRACT</b>	<b>6</b>
<b>CHAPTER ONE</b>	<b>6</b>
<b>INTRODUCTION</b>	<b>6</b>
1.1 Background Study	6
1.2 Problem Statement	7
1.3 Research Question	8
1.4 Objective	8
1.5 Scope	8
1.6 Significance of the Study	9
1.7 Contribution to Social Business	10
1.8 Thesis Structure	10
1.9 Summary	11
<b>CHAPTER TWO</b>	<b>12</b>
<b>LITERATURE REVIEW</b>	<b>12</b>
2.1 Overview of Phishing Detection Techniques	12
2.1.1 Traditional Methods for Phishing Detection	13
2.1.1.1 Blacklist Checking	13
2.1.1.2 Heuristic-Based Approaches	13
2.1.2 Integration of Machine Learning Techniques in Phishing Detection	13
2.2 Role of Natural Language Processing in Cybersecurity	14
2.2.1 Understanding and Processing User-Generated Content for Security Purposes	14
2.2.2 NLP Techniques Relevant to Phishing Detection	15
2.3 Advancements in Deep Learning for Security Applications	15
2.3.1 Recent Advancements in Deep Learning Applicable to Cybersecurity	15
2.3.2 Deep Learning Architectures for Phishing Detection	15
2.4 Challenges and Gaps in Current Research	16
2.4.1 Limitations of Existing Phishing Detection Systems	16
2.4.2 Addressing Challenges with NLP and Deep Learning	16
Summary of Literature Review	16
<b>CHAPTER THREE</b>	<b>21</b>
<b>METHODOLOGY</b>	<b>21</b>
3.1 System Development Methodology	21

3.2 System Architecture Design	24
3.2.1 Presentation Layer	25
3.2.2 Application Layer (Business Logic)	26
3.2.3 Data Layer (Database and ORM Integration)	27
3.3 Diagrams	29
3.3.1 Use Case Diagram	29
3.3.2 Swimlane Diagram:	31
3.3.3 Sequence Diagram :	34
3.4 System Requirements	39
3.4.1 Requirement Identification	39
3.4.2 Functional Requirements	39
3.4.3 Non-Functional Requirements	40
<b>CHAPTER FOUR</b>	<b>42</b>
<b>RESULT AND DISCUSSION</b>	<b>42</b>
4.1 Purpose of the System	42
4.2 User Interface Modules	44
4.2.1 Login Page	44
4.2.2 Registration Page	45
4.2.3 Home Page (Landing Page)	46
4.2.4 Dashboard	47
4.2.5 Email Check Module	48
4.2.6 URL Check Module	49
4.2.7 Admin Panel	50
4.2.4 System Integration	51
4.2.4.1 API Gateway: Frontend to Backend Communication	52
4.2.4.2 Database Integration: User Management, Predictions, and Feedback Logs	52
4.3 Model Development and Workflow	53
4.3.1 Hybrid Model Architecture	54
4.3.2 Data Flow Pipeline	55
4.3.3 Feedback Loop for Continuous Improvement	56
4.4 Testing	56
4.4.1 Unit Testing	56
4.4.2 System Testing	59
4.4.3 User Testing	61
<b>CHAPTER FIVE</b>	<b>64</b>
<b>CONCLUSION AND FUTURE WORK</b>	<b>64</b>
<b>6 References :</b>	<b>66</b>
<b>7. appendix</b>	<b>69</b>
Gantt Chart	69

## LIST OF TABLES

Table 1.1: Literature Review	19
Table 1.2: Functional Requirements	39
Table 1.3: Non-Functional Requirements	40
Table 1.4: Unit Testing	57
Table 1.5: System Testing	59
Table 1.6: User Testing	62

## LIST OF FIGURES

Figure 1.1: Incremental Development Model	24
Figure 1.2: Layered architecture	25
Figure 1.3: Use Case	30
Figure 1.4: Swimlane	32
Figure 1.5: Sequence Diagram	35
Figure 1.6: Activity diagram	37
Figure 1.7: log-in page	45
Figure 1.8: Registration page	46
Figure 1.9: Home page	47
Figure 1.10: Dashboard	48
Figure 1.11 and 1.12: Email Check Module	49
Figure 1.13 and 1.14: URLCheck Module	50
Figure 1.15: Admin Panel	51
Figure 1.16: Model Development and Workflow	54
Figure 1.17 and 1.18 : User Feedback	62

## LIST OF ABBREVIATIONS

<b>Abbreviation</b>	<b>Full Term</b>
NLP	Natural Language Processing
DL	Deep Learning
ML	Machine Learning
URL	Uniform Resource Locator
GUI	Graphical User Interface
AI	Artificial Intelligence
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
SQL	Structured Query Language
ORM	Object-Relational Mapping
API	Application Programming Interface
DB	Database
SDG	Sustainable Development Goal
UI	User Interface
UX	User Experience
GDPR	General Data Protection Regulation

## **ABSTRACT**

This project presents PhishGuard, the real-time phishing identification tool designed with the help of the Natural Language Processing (NLP) and Deep Learning (DL) technologies. The system has been planned to identify malicious emails and URLs with a high level of accuracy due to the combination of advantages of Convolutional Neural Networks (CNN), Bidirectional Long Short-Term memory (BiLSTM), and feature extraction based on the TF-IDF. PhishGuard was constructed in modular and layered architecture and was created with an Incremental development model, which integrates a user interface designed using the Flask framework, secure authentication, prediction API, and a feedback system to continuously enhance model performance. Focusing on a real-life implementation, the system guarantees the responsive user experience and safe data manipulation through SQLite and SQLAlchemy ORM. In addition to delivering technical performance, the project helps achieve Sustainable Development Goal (SDG) 16: Peace, Justice, and Strong Institutions by decreasing the number of cybercrimes, maintaining the trust in the system, and improving the adherence to the data protection rules. The PhishGuard P2P will eventually have transformed individuals and organizations into having a safe platform to socialize online, promote ethical online governance and institutional robustness.



# CHAPTER ONE

## INTRODUCTION

### 1.1 Background Study

Online phishing attacks present a serious danger in the modern age, targeting individuals and organizations to steal sensitive information. Traditional phishing defenses examine URL patterns and email headers through heuristics to spot phishing attempts. Standard phishing detection systems detect basic phishing attempts but fail to identify new method types and advanced evasive techniques used by cyber attackers. Online hackers alter email content to disguise their tactics and use social manipulation to fool users making basic detection systems unable to stop them. Modern detection methods need to advance because current defense systems no longer protect users from evolving phishing attacks.

New advances in NLP and DL technologies display promising results in better detecting phishing threats. The analysis of email wording by NLP tools detects phishing patterns using both writing structure and context understanding (Peng et al., 2018). Phishing emails contain specific warning signs like deceptive language, urgent requests and suspicious links which can help security systems detect threats. NLP-based models detect phishing attempts with improved accuracy because they can process new types of data and defend against new tactics.

Phishing detection gets better when deep learning models such as CNNs and BiLSTM extract detailed features from large data collections (Salloum et al., 2022). BiLSTM analyzes the timing of email content to see how it flows through messages while CNNs find patterns of email content structure. GCNs represent a new approach to phishing detection which examines how words connect in email texts to provide better results than conventional methods (Alhogail & Alsabih, 2021).

The field still faces problems in reaching the best possible detection results. The ability of phishing detection systems to track new attacks poses one of the main challenges to security. Using NLP and DL together in phishing detection systems lets us build better solutions that detect new threats better while keeping false alarms low. Research teams need to improve and strengthen these models so they can work better in actual practice.

## 1.2 Problem Statement

Phishing attacks keep changing and developers face difficulty with current security systems to detect these attacks. The main problem for phishing detection lies in making the system flexible enough. Attackers modify email contents and sender details while replacing URLs to bypass static detection models (Verma & Shashidhar, 2015). Simple detection solutions cannot match advanced phishing tactics because they need fixed patterns to work. Modern phishing detectors need dynamic features to stop new attack types uncovered by Alhogail & Alsabih (2021).

Developing effective phishing detection becomes challenging because current testing and training resources remain limited. Our machine learning and deep learning models require accurate datasets to detect phishing attacks across multiple platforms (Peng et al., 2018). Our efforts to obtain properly classified phishing examples continue to face significant obstacles. A limited phishing method collection section impacts model capability to detect new tactics and triggers unwanted safety notices to everyone.

Phishing defense systems fail too often when they flag genuine websites as fake. When detection systems produce too many wrong alarms users get angry and doubt their security system which makes them dismiss important warnings (Alhogail & Alsabih, 2021). The problem impacts companies using automatic tools to spot phishing attempts especially when they protect important data. A system to identify phishing attacks effectively needs advanced feature engineering plus model optimization to reduce wrong detections while keeping good detection rates.

Deep learning models present an extreme challenge due to their demanding computation needs. Real-time email analysis systems demand fast system processing to review high traffic volumes of incoming emails (Yang et al., 2019). The big demands deep learning algorithms place on computer systems make them difficult to use in situations with scarce processing power.

Creating better phishing detection systems that adjust to new threats and handle data effectively while remaining easy for users to understand remains our highest priority. Future studies must examine better ways to adapt detection systems while making more robust training data available and improving performance speed for practical phishing protection designs (Peng et al., 2018).

## 1.3 Research Question

- What are the existing studies and methodologies for detecting phishing attacks, and what are their limitations?

- How can a hybrid model combining NLP and Deep Learning techniques be designed to enhance phishing detection?
- How does the proposed hybrid model perform compared to existing approaches in detecting phishing attempts?

## **1.4 Objective**

The primary objectives of this research are:

- To investigate current studies and methodologies related to phishing detection, focusing on the use of NLP and DL techniques to identify existing gaps and challenges.
- To propose a new hybrid model that integrates NLP for advanced textual analysis and DL for automated feature extraction, providing an enhanced phishing detection framework.
- To evaluate the performance of the proposed model using various benchmark datasets, considering key performance indicators such as accuracy, precision, recall, and false positive rates.

## **1.5 Scope**

This research aims to:

- Develop a hybrid phishing detection model that leverages NLP for textual analysis and DL for feature extraction.
- Evaluate the model's performance using diverse datasets, assessing metrics such as accuracy, precision, recall, and false positive rates.
- Analyze the model's adaptability to new phishing tactics by testing against emerging phishing datasets.
- Propose strategies to mitigate computational challenges, facilitating the deployment of the model in real-time environments.

## **1.6 Significance of the Study**

This research shows how it could transform security technology by building better systems to spot phishing attempts. Better security tools become essential because phishing attacks grow stronger and more difficult to detect. Our research helps create new and better ways to protect against cyber threats through NLP and Deep Learning technologies.

### **Benefits to Individuals**

Most phishing threats target specific people who want to steal their private information including passwords, bank accounts, and social security numbers. Our phishing detection system helps stop cybercriminals from stealing information that would harm users both financially and mentally. Our system utilizes NLP and DL technology to analyze email data better while finding suspicious activities which improves the safety of online experiences. A better phishing detection system helps users feel more secure when they make digital transactions online.

### **Benefits to Organizations**

Organizations face major hazards from phishing attacks because these threats can break into their data, cost them money, hurt their brand value and force them to pay fines. Cybercriminals use personal attacks to steal sensitive company data by sending special emails to employees. An excellent phishing protection solution lets organizations secure their resources while keeping customer trust and meeting legal requirements. Adaptive security detection helps protect businesses from advanced phishing threats and stops operational problems caused by cyber breaches.

### **Contribution to Research and Development**

This work creates research direction while offering basic ideas for improving cybersecurity by developing a hybrid phishing detection system. The research outcomes can help expert teams design better phishing detection systems while finding new ways to work and fix present system issues. The research supports the development of AI security systems through its effort to stay ahead of phishing defense.

## **1.7 Contribution to Social Business**

The importance of the research to SDG 16: Peace, Justice, and Strong Institutions is that it will boost digital security, limiting the effects of cybercrime, especially the phishing crime. Phishing is an act that destroys the confidence of the digital system and gives substantial risks to people, firms, and organizations. This project can help shore up cybersecurity infrastructure that is required to maintain secure and trusted digital environments by creating an effective phishing detection system with the assistance of Natural Language Processing (NLP) and Deep Learning (DL) tools. It enhances responsibility, openness and user trust during online interaction. Furthermore, the system can help to accommodate the international data protection regulations, including GDPR and HIPAA, which enhance ethical governance and secure internet involvement. As a result of these activities, the research will promote stronger institutions, a more secure, and fairer digital community.

## **1.8 Thesis Structure**

Our research follows several parts which explain all aspects beginning with its purpose and continuing through the methods used and results delivered. The organization of the thesis is as follows:

### **Chapter 1: Introduction**

This chapter outlines our study methods and introduces readers to the topic of phishing attacks and their harmful effects on people and businesses. The paper explains our research project and shows how it focuses on social business and SDGs while setting out our problems to resolve along with research goals and studies.

### **Chapter 2: Literature Review**

Our literature reviews deep dives into past studies on phishing detection with special attention on existing heuristic-based and machine learning methods plus deep learning approaches. The research examines how Natural Language Processing (NLP) finds phishing content and points out missing elements in today's detection methods. This chapter evaluates published research articles about cyber threats while monitoring today's phishing patterns and studying established cybersecurity standards.

### **Chapter 3: Research Methodology**

This chapter outlines how the team created a phishing detection system using chosen procedures and methods. It explains all steps of data acquisition and processing followed by model development and evaluation criteria selection. The chapter outlines the setup of our experiments including deep learning methods and NLP technology tools.

### **Chapter 4: System Development and Implementation**

This chapter demonstrates how to put our combined phishing detection system into operation. The section outlines how to build and operate our phishing detection solution including design creation, model learning and refining plus deployment techniques. The chapter reviews the technical complications of deployment along with how these were resolved.

### **Chapter 5: Results and Discussion**

Our research chapter explains the testing outcomes of our phishing detection model. Our evaluation compares standard performance indicators including accuracy and false positives with current detection systems. The study team examines its analysis results using research targets as benchmarks to propose updates.

## **1.9 Summary**

Phishing is an emerging cybersecurity problem because cybercriminals, through misleading messages and counterfeit websites, take advantage of sensitive personal and organisational information. The obsolete heuristic based methods of detection can no longer be used to counter the growing advanced strategies being used by the attackers. To this end, this study proposes the use of a hybrid phishing detection mechanism, a system, named PhishGuard, that utilizes both the Natural Language Processing (NLP) and Deep Learning (DL) framework to enhance the detection of threats, minimize the occurrence of false positives, and adapt to new tricks developed by phishers. The system attains high accuracy, precision, recall, and false positive rate on various benchmark datasets by considering the feature extraction of complex semantic structure of email and DL to learn semantics of email according to NLP.

This project will directly adapt to Sustainable Development Goal (SDG) 16: Peace, Justice and Strong Institutions: the project will also involve the promotion of digital safety, the reduction of cybercrime, and the development of online trust in institutions that will help steer SDG 16: Peace, Justice and Strong Institutions. Through the safety of digital communications channels and preserving integrity of sensitive information, the system promotes ethical digital governance and increased trust to institutional technology. It allows people, companies and government agencies to react to cyber threats in a more effective manner, buttressing the pillars of safe and transparent cyber-space.

## **CHAPTER TWO**

### **LITERATURE REVIEW**

#### **2.1 Overview of Phishing Detection Techniques**

Phishing detection models now use advanced techniques instead of basic blacklist and heuristic scans. Under these traditional approaches rules were pre-set but over time proved ineffective due to evolving phishing tactics. The study by Hussein et al. (2022) shows that systems using blacklists track only existing threats but generate many false alarms when using heuristic detection methods that rely on fixed rules. Researchers now use ML and NLP to build enhanced tools for phishing identification. Using machine learning technology we can analyze email texts alongside URLs and metadata details to enhance our detection methods. Strong models that combine SVM-NLP with Probabilistic neural networks show promise for advanced phishing detection outcomes through a sophisticated multiclass learning system that reduces incorrect flagging (Kumar et al., 2020). The combination of CNNs and BERT feature extraction outperforms previous detection systems for spotting phishing URLs effectively and correctly as shown in Elsadig et al. 2022. Phishing attacks use personal weaknesses to trick people into sharing confidential data like usernames, passwords, credit cards, and private information. Technology experts have built different systems for finding phishing scams since the start with simple rules and modern artificial intelligence tools. Volatility in these methods depends on their ability to monitor ongoing phishing attacks while keeping identified threats and false findings to a minimum. The standard methods stick to basic rules but new approaches depend on AI to boost their ability to detect threats and adjust to new situations (Korkmaz et al., 2020).

### **2.1.1 Traditional Methods for Phishing Detection**

#### **2.1.1.1 Blacklist Checking**

Security systems built on blacklists operate by blocking URLs found in lists managed by Google Safe Browsing, PhishTank, and APWG. The organizations keep these lists current as part of their efforts to block phishing site access. This method lets you deploy defense systems easily because it keeps things simple. Users find their access to blacklisted URLs either blocked or they receive notification upon trying to access them. As a prevention method, blacklisting cannot protect users from attacks that emerged since the last database update. The effectiveness of blacklists breaks down according to Colhak et al. (2024) when attackers adjust their URLs and website content by just a few details. The continuous evolution of phishing attacks makes it hard to keep a complete blacklist current (Aassal et al., 2020).

### **2.1.1.2 Heuristic-Based Approaches**

Pre-defined rules and patterns enable heuristic-based phishing detection systems to identify abnormal content in URLs, web pages, and emails. We analyze digital indicators like multiple subdomains on a single URL, spelling mistakes, abnormal address lengths, domain name replacement with IP addresses, and keywords that signal phishing attempts. Heuristic methods scan HTML content and metadata for hidden form fields and detect obfuscated scripts alongside excessive redirects (Korkmaz et al., 2020). These approaches successfully identify phishing sites through their analysis of URL patterns without requiring blocking lists. The approach frequently marks legitimate websites as malicious because phishing sites and genuine websites can match too many common features. Updates to heuristic rules must occur often since attackers keep developing new ways to execute phishing attacks (Aassal et al., 2020).

### **2.1.2 Integration of Machine Learning Techniques in Phishing Detection**

Machine learning now helps detect phishing attacks better thanks to improved detection methods. Machine learning algorithms evaluate multiple details in URLs and websites plus network behavior to detect phishing attacks better. The detection system incorporates three successful ML approaches for better phishing detection: supervised, unsupervised, and reinforcement learning algorithms as described by Colhak et al. (2024). The key benefit of using machine learning for phishing detection is its ability to spot new attacks through data pattern analysis from big data samples. ML systems detect phishing attacks by analyzing special text sequences in URLs paired with WHOIS registration data and webpage structure components (Aassal et al., 2020). The research by Korkmaz et al. (2020) shows that Decision Trees, Random Forest, Support Vector Machines and CNN as well as BERT model types effectively identify phishing attacks. ML-based solutions deliver strong performance but encounter practical difficulties. We face a crucial barrier in obtaining sufficient large and current data sets needed to build effective models. Specific ML systems remain at risk of being defeated when attackers intentionally manipulate website elements to hide their actions. The heavy processing requirements of advanced ML models make them ineffective for networks with low computational capacity according to Colhak et al., 2024.

## **2.2 Role of Natural Language Processing in Cybersecurity**



Natural Language Processing serves as a key cybersecurity solution because it automatically analyzes and makes sense of content users create. NLP examines vast amounts of text-based data to find security threats including phishing emails and social engineering attacks plus fake news. NLP analyzes text in different communication platforms to spot signs of cyber threats (Koide et al., 2024).

### **2.2.1 Understanding and Processing User-Generated Content for Security Purposes**

NLP detects security dangers in user content by examining social media posts and customer feedback through its analysis methods. We use methods including tokenization, named entity recognition and sentiment analysis to find important details and sort text content into safe or dangerous types. NLP technology detects phishing attempts through these methods by recognizing signs of social engineering attacks in writing. Security systems use NLP to look at phishing email contexts to tell genuine content from dangerous content more accurately (Alhogail & Alsabih, 2021).

### **2.2.2 NLP Techniques Relevant to Phishing Detection**

NLP techniques that help find phishing show great results in sentiment analysis, text classification, and topic modeling. Support vector machines and deep learning models work together with text classification to find phishing content in emails while sentiment analysis looks for dangerous emotion in messages. The TF-IDF and word embedding methods help phishing detection systems work better by recognizing specific patterns found in these emails according to Kumar et al. (2020). Cybersecurity systems gain better protection against phishing attacks through NLP methods such as LSI. Task response varies based on email content.

## **2.3 Advancements in Deep Learning for Security Applications**

Deep learning technology helps cybersecurity professionals detect threats using its abilities to find patterns and detect unusual activities automatically. New deep learning techniques deliver effective tools to spot phishing attempts, block malware and prevent unauthorized access to systems (Benavides-Astudillo et al., 2023).

### **2.3.1 Recent Advancements in Deep Learning Applicable to Cybersecurity**

Researchers now use advanced DL techniques including transformer models with attention mechanisms extended through hybrid approaches using multiple neural network architectures to boost detection results. BiLSTM and CNN work together to detect phishing attacks effectively by extracting time-based and location-based features at once according to Ozcan et al.'s 2021 research. Pre-trained models and transfer learning helps cybersecurity systems update themselves and fight new threats very fast with little need for retraining according to Zhang et al. 2020.

### **2.3.2 Deep Learning Architectures for Phishing Detection**

The deep learning frameworks CNNs and RNNs demonstrate strong ability to identify phishing threats. CNNs detect design features in phishing web pages while RNNs with LSTM and GRU units analyze email content and URLs effectively according to Aljofey et al. 2020. Deep neural networks enhanced with LSTM components for long short-term memory help track phishing URLs through local connections and long-distance dependent relationships (Ozcan et al., 2021). Deep learning shows how it can transform cybersecurity by detecting cyber threats as they happen and adapt to new patterns.

## **2.4 Challenges and Gaps in Current Research**

### **2.4.1 Limitations of Existing Phishing Detection Systems**

Current phishing detection systems have ongoing technical constraints even with their recent improvements. Researchers face a major problem of identifying too many legitimate websites as phishing sites which negatively impacts how users experience these websites (Elsadig et al., 2022). The present approach to detection fails to identify unknown attacks since it depends on past incident data and familiar patterns to operate (Maneriker et al., 2021). Poor availability of current phishing datasets makes it hard for detection models to reliably spot different attack types (Shirazi et al., 2020). Running deep learning models on mobile devices proves difficulty because of their limited power and battery resources (Haynes et al., 2021). Current detection methods do not perform well on phishing content written in different languages because most training data contains only English texts.

### **2.4.2 Addressing Challenges with NLP and Deep Learning**

The latest advances in phishing detection systems have failed to overcome their basic technical problems. Researchers identify too many honest websites as phishing locations which harms user experience when dealing with real websites. The current detection system fails to detect new attacks as it relies on previous incident records and standardized patterns to perform its tasks (Maneriker et al., 2021). Detecting different attack types becomes challenging for detection models due to insufficient access to modern phishing datasets.(Shirazi et al., 2020) Mobile devices have insufficient CPU power and battery capacity which makes deep learning model execution challenging. (Haynes et al., 2021). Most phishing detection systems have trouble identifying foreign language phishing content since their training data uses only English texts.

#### **Summary of Literature Review**

The analysis reveals that researchers have developed phishing detection from basic blacklists and heuristic rules into complex machine learning and deep learning solutions. Existing blacklist detection tools work well against established phishing attacks because they use identified threats in database files. The flexible nature of heuristic detection methods leads to many incorrect results that hurt their trustworthiness.

Machine learning now detects phishing with high accuracy by using information from URLs, website content, and metadata through modern learning methods. SVMs and decision trees show strong results as supervised learning models but deep learning methods such as CNNs and transformer models achieve superior results in finding complex data patterns. The rapid progress in phishing detection technologies faces multiple hurdles including scarce good-quality data, massive processing demands, and threats from cyber-attacks.

Natural Language Processing (NLP) examines written material to identify phishing characteristics in both email messages and web pages. Research shows that text analysis approaches such as sentiment analysis, topic modeling, and text classification can detect phishing better than before. Deep learning tools now include new hybrid systems consisting of CNNs and recurrent networks to detect phishing activity better.

Table 1.1: Literature Review

Title of the Paper	Author and Year	Method	Research Gap
A novel hybrid approach of SVM combined with NLP and probabilistic neural network for email phishing	Kumar, A., Chatterjee, J. M., & Díaz, V. G. (2020)	In this research study, SVM, NLP, and PNN are used for phishing detection. It uses message and tokenization to extract features, deletes stop words and performs a rigid and elastic classification for achieving high accuracy on 1705 inbox data.	The current approaches have no provision for dynamic adaptation and the presence of multiple classifiers. To overcome this, the paper brings out the hybrid SVM-PNN with enhanced feature selection and better accuracy.
Intelligent Deep Machine Learning Cyber Phishing URL Detection Based on BERT Features Extraction	Elsadig, M., Ibrahim, A.O., Basheer, S., Alohal, M.A., Alshunaifi, S., Alqahtani, H., Alharbi, N. and Nagmeldin, W., 2022.	For feature extraction, this paper employs BERT and as a phishing URL detection technique, a deep learning technique known as Convolutional Neural Networks (CNNs) is applied. Featuring	Since they are not very effective in feature extraction, conventional techniques fail in the accurate identification of the phishing URLs because of their complexities. This

		NLP to the content, the dataset contains 472,259 entries. It is integrated into a recommendation list, which has an accuracy of 96.66%.	kind of research question is solved by this study with the help of BERT with respect to classic ML for the effective feature extraction.
Detection of Phishing Websites by Using Machine Learning-Based URL Analysis	Mehmet Korkmaz, Ozgur Koray Sahingoz, Banu Diri (2020)	Machine learning algorithms analyze URL features to detect phishing without relying on third-party services.	Existing methods lack real-time detection and adaptability to evolving phishing tactics.
Email phishing: Text classification using natural language processing	PPriyanka Verma, Anjali Goyal, Yogita Gigras, <b>2020</b>	Preprocessing of text data in the study involves tokenization, stemming, and stop-word removal from the textual data. Other common classifiers such as Support Vector Machine (SVM), Decision Trees, Random Forest, and Naive Bayes are used in this study to categorize phishing emails. These	Previous approaches to detection had poor preprocessing and time consuming classification for phishing mails. This study redresses this default by adopting NLP and multiple ML classifiers to increase the accuracy and to accommodate such a rich structured and unstructured text data.

		messages (ham/spam) contain 5574 messages, successfully getting an accuracy up to 98.77% by applying SVM.	
Phishing Website Detection through Multi-Model Analysis of HTML Content	Furkan Çolhak, Mert İlhan Ecevit, Bilal Emir Uçar, Reiner Creutzburg, Hasan Dağ (2024)	A fusion model combining MLP and NLP models (CANINE and RoBERTa) to analyze HTML content for phishing detection. The approach achieves high accuracy and F1-score through feature extraction and model fusion.	Limited availability of recent datasets and lack of exclusive focus on HTML content in existing studies. The study addresses this by creating a new dataset and introducing a novel model fusion technique.
An In-Depth Benchmarking and Evaluation of Phishing Detection Research for Security Needs	Ayman El Aassal, Shahryar Baki, Avisha Das, Rakesh M. Verma (2020)	Introduces 'PhishBench,' a benchmarking framework to evaluate phishing detection methods using a unified system and diverse datasets.	Existing methods lack standardized evaluation frameworks, leading to inconsistencies in assessing phishing detection techniques.

## **CHAPTER THREE**

### **METHODOLOGY**

#### **3.1 System Development Methodology**

##### **3.1.1 Incremental Development Model:**

Development of the phishing detection system followed the Incremental Development Model that supported partitioning and modular implementation, as well as gradual integration. First, the system architecture was structured in such a way that the layers are well-separated (presentation layer (UI), application layer (Flask backend), and data layer (SQLite with SQLAlchemy ORM)) model. Such basic modules as user registration, user log in, and the fundamental interface were created and tested independently in order to form a solid base.

Future updates involved installing more important features such as email and URL phishing security, based on deep learning and machine learning systems, in later versions. AJAX-based communication was used to integrate these modules to the backend seamlessly, hence providing real-time feedback without the need of page-reload. The two models which are the hybrid model

trained independently and the validated model were implemented under the application logic since they provide realistic and quick predictions.

The last increment was dedicated to the improvement of the system usability and control by administration. Such functionality as user feedback submission, the management of the Inbox of the administrator, and the future support of the models were introduced. Work on each step was then preceded by specific test and optimization, so that the system could be constantly improved without losing its scalability, security, and convenience to the user.



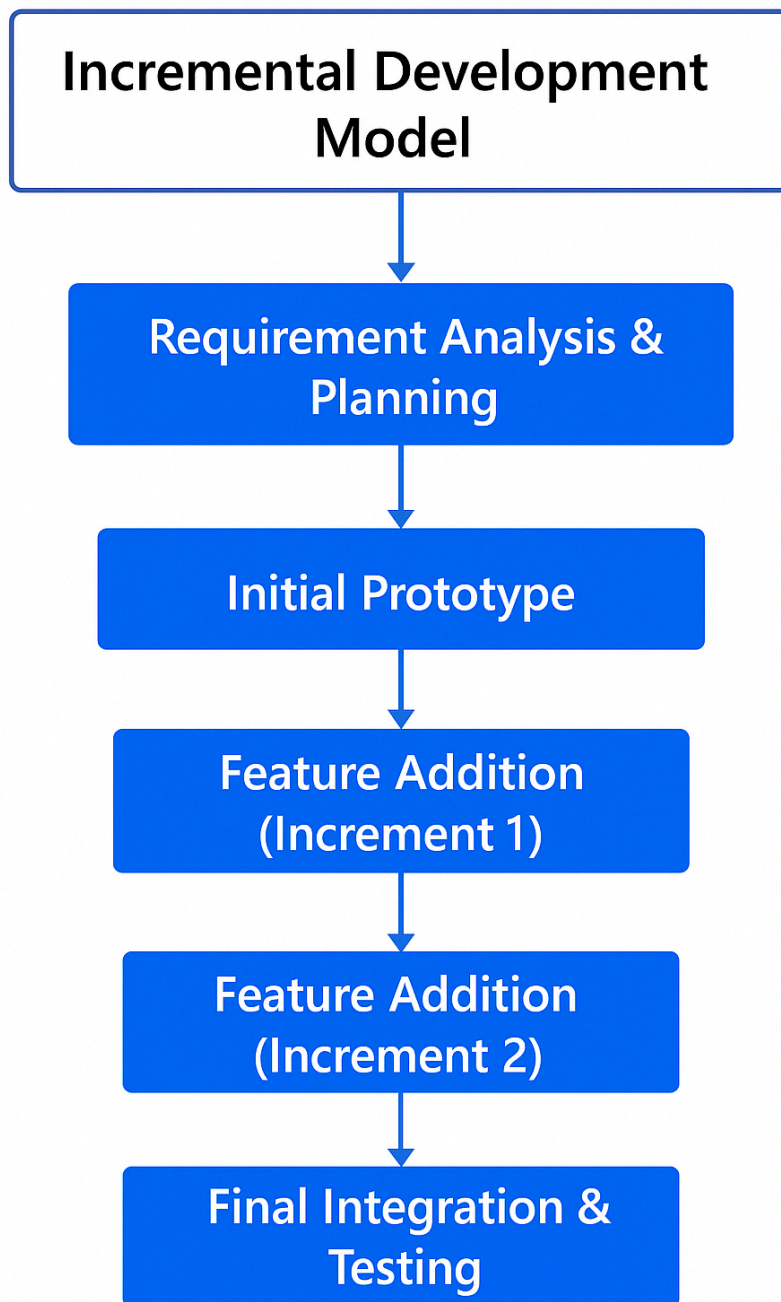


Figure 1.1: Incremental Development Model

### 3.2 System Architecture Design

#### PhishGuard - Layered Architecture

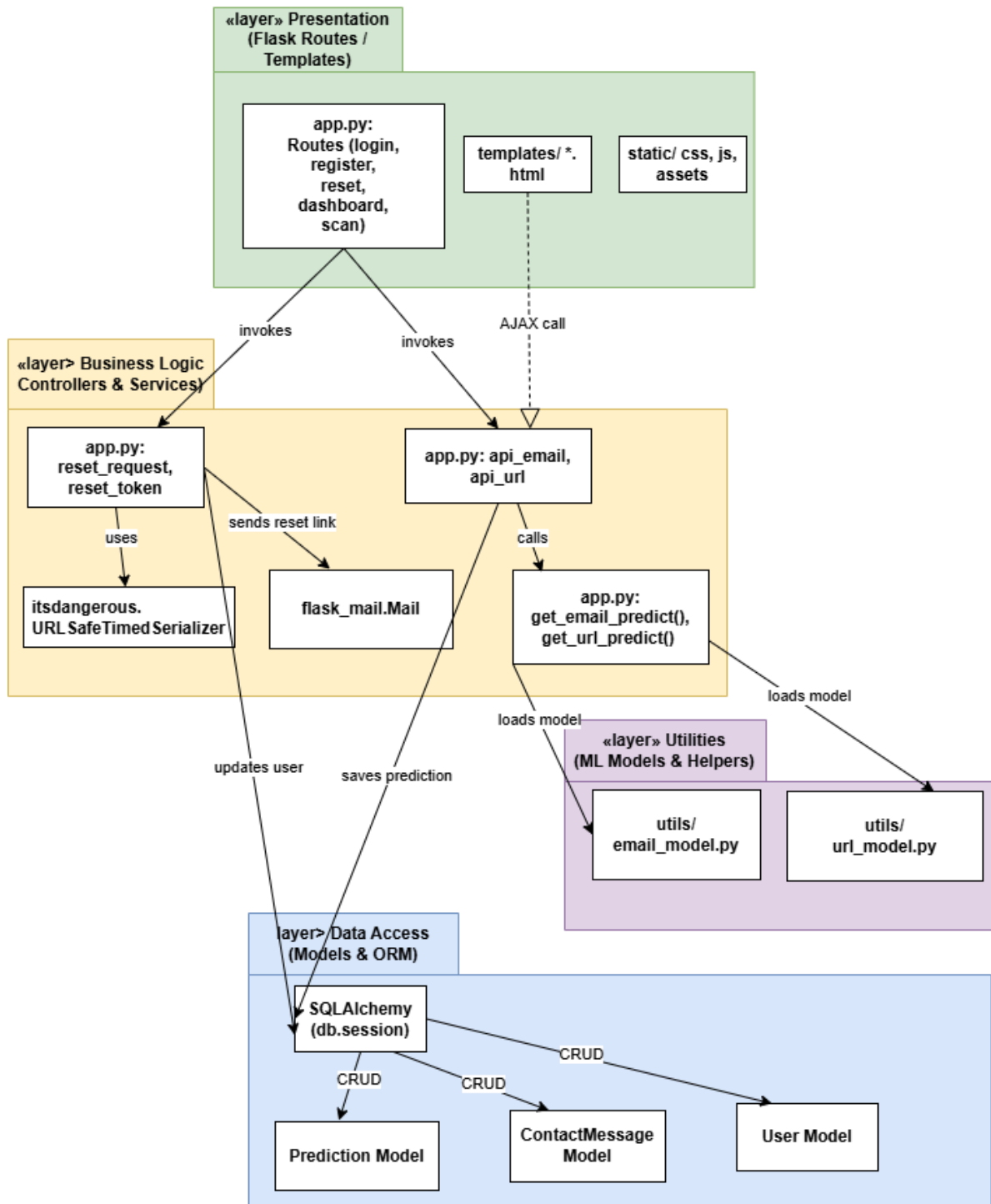


Figure 1.2: Layered architecture

The PhishGuard system follows a **three-layered architecture**, which logically separates responsibilities into:

### 3.2.1 Presentation Layer

The system PhishGuard Presentation Layer is the main interface that the users communicate with the application. It was created on the Flask web framework and is based on the Jinja2 templating engine along with HTML, CSS, JavaScript and Bootstrap 5 to provide a modern, responsive and user friendly experience. This layer has the responsibility of displaying all the components on the client side, receiving all user inputs and also maintaining smooth communication with the backend.

There are a few important pages on the interface that include a login (/login), registration (/register), email scanner (/email), URL scanner (/url), dashboard (/dashboard), and contact form (/contact). All pages are coded in HTML as a structural layout, designed with CSS and Bootstrap 5 as a visual appearance to match and be responsive across devices and refined with JavaScript and AJAX to support dynamic changes in the content. An example is when a user sends an email file or a URL link to be evaluated, the data will be sent via AJAX in an asynchronous parameter to the server side end points (/api/email or /api/url). After processing, the backend gives a result in prediction which is then displayed dynamically on the page without loading an entire page.

Variously speaking, the interface should be usable and friendly to users of different levels of technical configurations. The presence of hint messages in real-time, user friendliness, layouts and navigation are some of the features that will create convenience in navigation and involvement of the viewer. In addition, one of the factors in the presentation layer is security. Jinja2 autoescape feature is used to reduce the number of cross-site scripting (XSS) exploits, session management and optional CSRF libraries are used to prevent unauthorized user actions.

In a nutshell, Presentation Layer becomes a medium that connects users and smart phishing detection services that are provided by PhishGuard. It can not only ensure an efficient

communication between the users, but also maintain the level of secure and responsive real-time threat assessment platform.

### **3.2.2 Application Layer (Business Logic)**

The Application Layer is the heart of the operational system PhishGuard, performing user interaction, implementation of the system logic, and maintaining communication between server interface and the database. The layer was created with the Flask web framework with the language Python and wraps business logic needed to perform authentication, execution of phishing predictions, user data management and provide API endpoints with dynamic actions.

The layer integrates several crucial frameworks and libraries to achieve durability and division. Flask forms the basis of routing, receiving and processing HTTP request and response as well as designing the overall application flow. The session management and authentication of users is done by using Flask-Login, secure password reset features are done using itsdangerous to generate the tokens and Flask-Mail to send emails. It uses SQLAlchemy as the Object-Relational Mapping (ORM) instrument to simplify communication between the database and the application.

All important routes and API endpoints are determined by the Application Layer. These are login, registration, and verification of emails and web addresses scanning, dashboard visualization, and contact inquiry forms. Decorators, `@login_required` and self-defined `@admin-required` are used to enforce access control so that limited resources can only be accessed by authorized users. Further, the system features a full password reset procedure that comprises tokens, password reset links, encryption through hashing passwords, and email reminders to the user.

`/api/email` and `/api/url` are the two major API endpoints where .eml files are uploaded and raw URLs are passed respectively. These endpoints implicitly call the respective deep learning

models through lazy loading (`get_email_predict()` and `get_url_predict()`) procedures, and then internally it calls utility modules to accomplish preprocessing and prediction. The results of each scan, including metadata associated with the scan (e.g. the confidence of a prediction, its type, and time of its creation) are registered in the database to facilitate the dashboard analytics and user history data.

As well as managing prediction workflows, the Application Layer provides additional endpoints, `/api/history` and `/api/stats`, that allow viewing historical data and displaying real-time system statistics both to individual users and to administrators. Entries to ensure data validation is strictly applied in order to avoid submission of invalid files with incomplete structures or invalid URLs and information privacy is adhered to through non-storage of raw email contents or sensitive user inputs.

In short, the Application Layer can be regarded as the brain of the PhishGuard platform that provides a combination of effective authentication, data validation, predictive modeling, and secure logging. Its modular and safe architecture guarantees its scalability and future system improvement.

### **3.2.3 Data Layer (Database and ORM Integration)**

The Data Layer of the PhishGuard system will deal with all persistent storage operations and will provide secure and efficient storing of such important data as user credentials, phishing prediction results, and contact messages. It is also SQLite, a lightweight relational database engine that is file-based and optimised to be used on small scale and local deployments, and SQLAlchemy, an ORM high-level library of Python under Object Relational mapping.

The reason we chose SQLite is that it is simple, portable, and setting up is easy especially in development and demonstration environments. The database file, `phishguard.db`, is located in the

system instance/- directory so it is easy to access and control the version. SQLAlchemy reduces the complexity of connecting both Python objects and database tables making manual usage of SQL queries unnecessary amongst providing concise and strong handling of transactions, schema and relationships.

The data schema of the system defines three major models. The User model stores the information related to the authentication of the user as follows: a unique identifier, email address, encrypted password (based on `werkzeug.security`), and flag of an administrative role (`is-admin`). At the Prediction model level we record every detection with a phishing attempt (user ID as foreign key to the User table, type of input (email or URL), final prediction verdict, confidence score, and a timestamp). Such a model allows monitoring and examining the user activity and model performance in the long-term. The third model, `ContactMessage`, records messages filled through the contact form, the name of the distiller, his email address, the body of his message and the time the user sent the message so that user inquiries can be well received by the user in order to respond accordingly.

The Data Layer is used to implement referential integrity via foreign keys constraints especially between the User and Prediction entities and as such data remain consistent and reliable which are stored. Moreover, the sensitive data (passwords) never gets stored in plain text form, on the contrary, it is first hashed and then stored to ensure the authentication mechanism is more secure and reliable.

The bottom line is that the Data Layer offers an appropriate environment to mitigate security as well as scale of all data driven activities on the PhishGuard system. It makes sure that user data, system logs, and even interaction are stored forever and readily available as the data is easy to analyze in real-time and to audit at later times.

### **3.3 Diagrams**

The overall structure and workflow of the sitting phishing detection system and the interdependency between different modules are described in the following diagrams.

### 3.3.1 Use Case Diagram

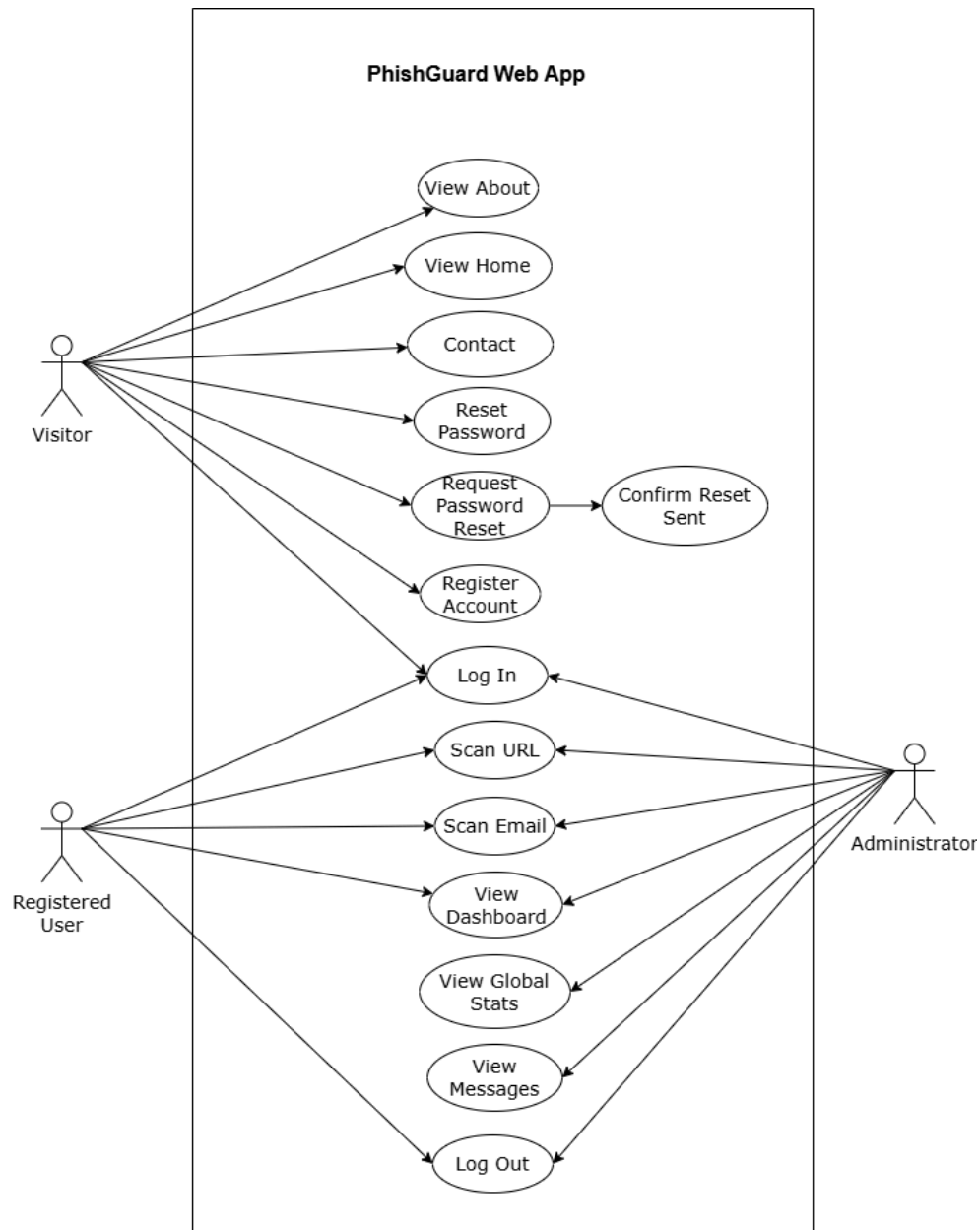


Figure 1.3: Use Case

The use case diagram, depicted in Figure 1.3, is a depiction of the communication between the different user roles and the main features of the PhishGuard web application. It is possible to

distinguish three main actors in the diagram: **Visitor, Registered User, and Administrator**, and each of these actors has various levels of access to the system according to his or her level of authentication and responsibility. With this diagram, the extent of features and user interactions can be determined, which is a basis of system behavior and interface expectations.

The visitors are unverified users who can view publicly-available features of the application. They can read the home page and the section of "About" that contains the general description of the aim and operation of PhishGuard. The visitors have an opportunity to contact the system administrators as well by filling in the contact form with a message. In case where a user has forgotten his or her password, visitors can visit the password reset page and enter a request to receive a link to reset the password, a confirmation message will then be shown, indicating that a request has been received. Moreover, the visitors are provided with the opportunity to create a new account and log in to obtain access to the safeguarded elements of the system.

Registered Users are granted privileges after authenticating and access sophisticated features. They are able to use the URL Scanner and Email Scanner tools, which enable them to identify the phishing threats in the submitted URLs or files uploaded with the format of .eml. There is also the ability of users to see their own unique dashboard and has scan histories and the other possibilities of interacting with the system. Moreover, the users can see the worldwide statistics that reveal a general assessment of the phishing activity throughout the platform, and also the messages that the administrative group sends to a user in response to a feedback or a question. Lastly, the Registered Users will have to use the system safely through logging out.

Administrators have certain functions in common with the Registered Users, including email viewing, scanning of URLs, viewing dashboard, getting global statistics and logging out. Nonetheless, the administrators have increased access and monitoring proficiencies. Specifically, they can access messages posted with the help of the contact form, which allows them to track the activity of the system, address the issues of concern to users, and handle feedback. They



equally access administrative functions by logging in at a secured portal embedded within the platform.

### 3.3.2 Swimlane Diagram:

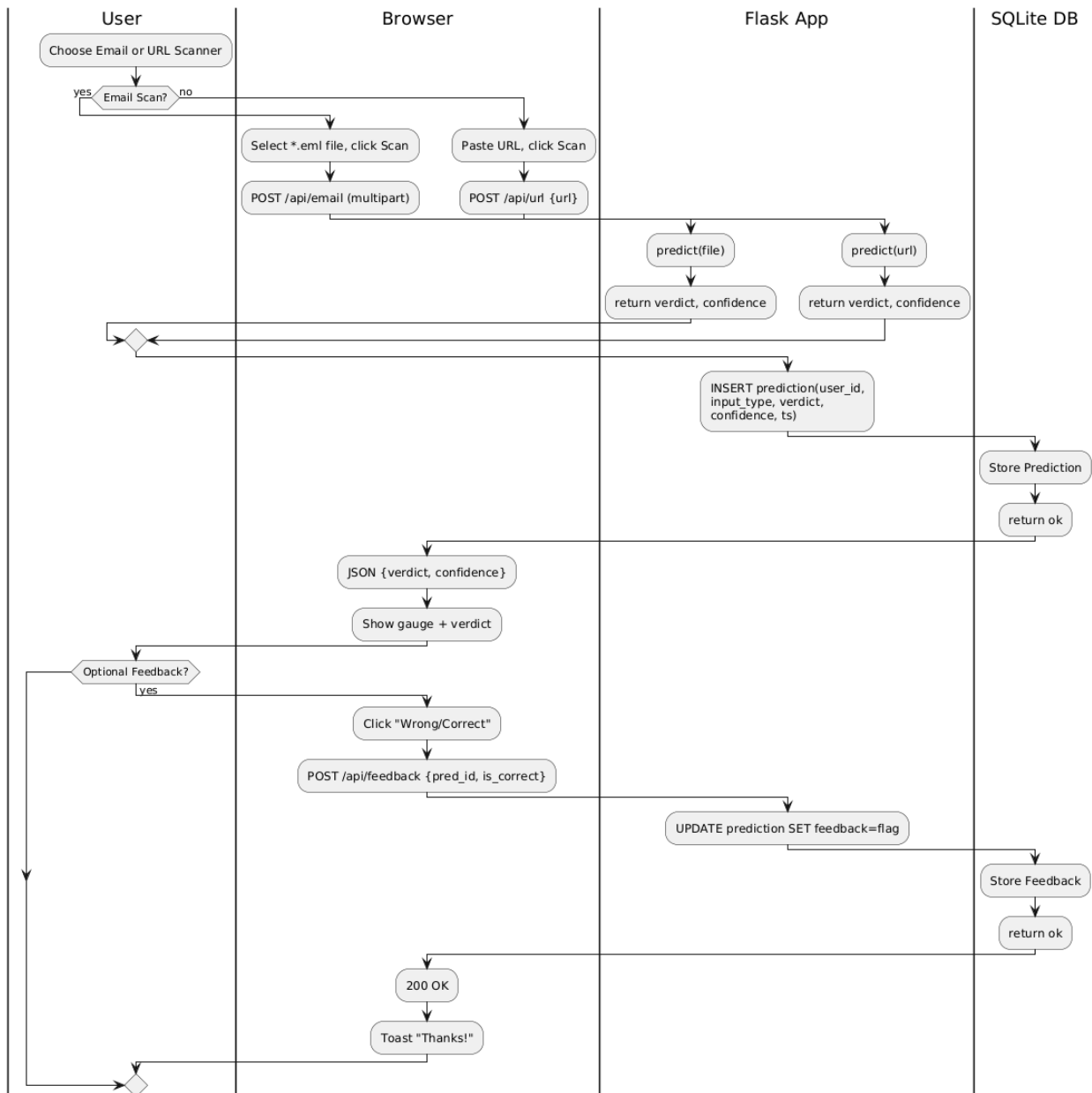


Figure 1.4: Swimlane

Figure 1.4 illustrates the user interaction workflow and back-end processing in the form of a Swimlane Diagram, showcasing the responsibilities and data flow across four major components: **User**, **Browser**, **Flask App**, and **SQLite Database**. The diagram highlights the complete lifecycle of a phishing detection request—starting from the user's selection to final prediction storage and optional feedback submission.

The process initiates when the **User** selects either the Email Scanner or URL Scanner feature. If the Email Scanner is chosen, the user uploads a .eml file; if the URL Scanner is selected, the user pastes a suspicious link. This user input is handled on the **Browser** side where the file or URL is transmitted through a POST request to the respective Flask API endpoint (/api/email or /api/url).

Upon receiving the input, the **Flask App** triggers its phishing prediction mechanism through the predict(file) or predict(url) functions. These functions utilize machine learning models to analyze the input and return two important results: the **verdict** (Phishing or Legitimate) and the **confidence score** indicating the model's certainty in its prediction.

After the creation of the prediction, the Flask App saves the result by performing SQL INSERT line in the SQLite Database with the user ID, input type, verdict, confidence score, and timestamp. The storage is confirmed by the database.

Then a JSON response is sent to the Browser, which includes the prediction results. It has a numerical icon (like gauge chart) and the per cent of the verdict, which the user interprets. The next step is that the system provides an option of giving feedback by the user when she feels that the answer was wrong or right.

In case the user wants to give a feedback, there is a pressing of a button labeled either Correct or Wrong. This provokes a feedback POST request to /api/feedback with the prediction id and correctness flag of the user. This feedback is processed by the Flask App and the record of the stored prediction in the DB is updated by adding a flag as feedback to the record using an

UPDATE query in the SQLite DB. Such extra feedback is stored and the response returned with a return code.

At last, the system sends the browser a confirmation (HTTP 200 OK) and the browser shows a toast message indicating success, like “Thanks!” indicating that the process is complete.

This swimlane diagram represents the combined effort of working toward front-end interactivity as well as back-end functionality (Flask-based), and backend logic persistence with the help of database, exposing the organized and efficient structure of PhishGuard in terms of real-time phishing detection and improving the model based on the feedbacks it receives.

### 3.3.3 Sequence Diagram :

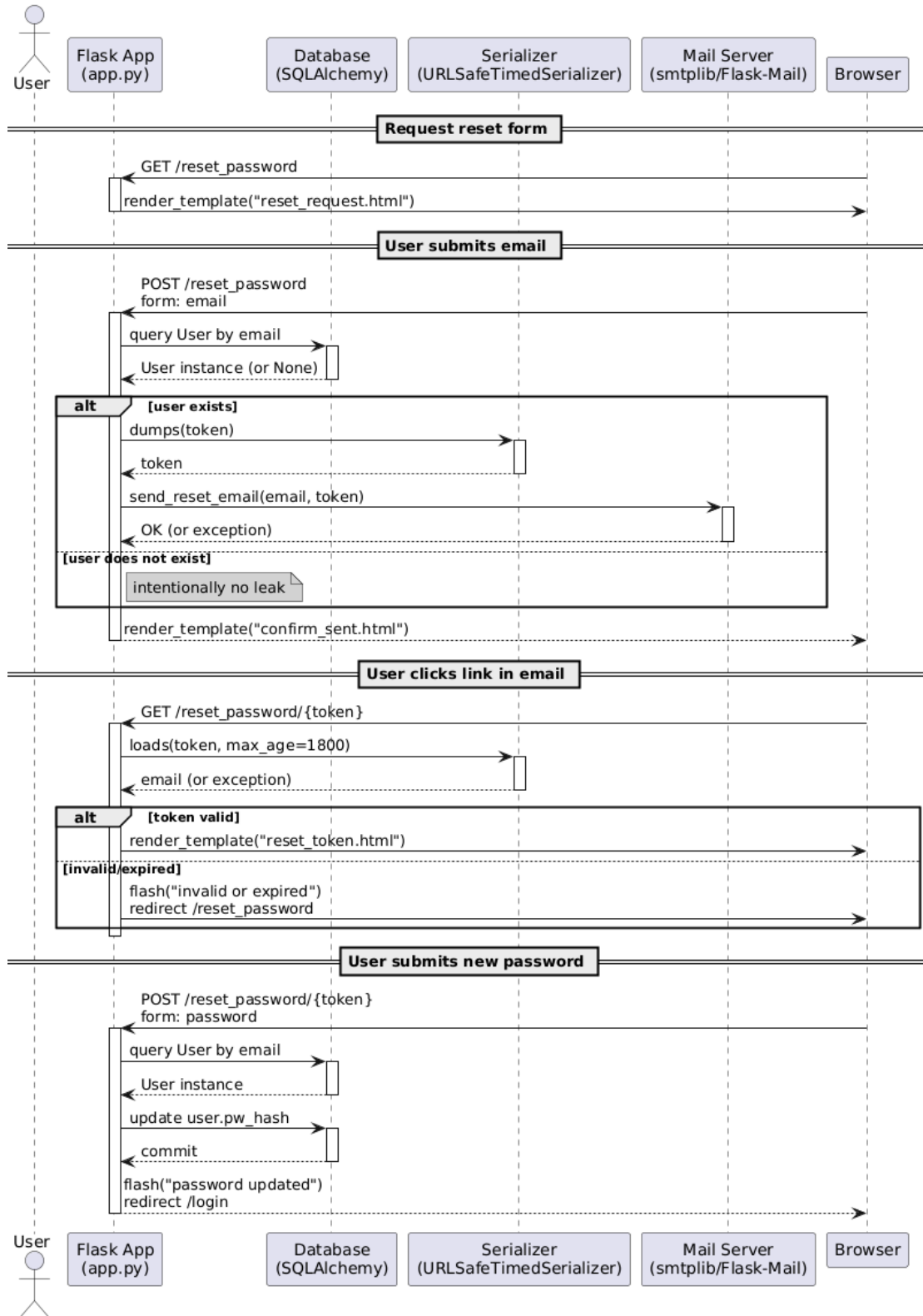


Figure 1.5: Sequence Diagram

Figure 1.5 displays the sequence diagram illustration of the password reset of the PhishGuard web application and how the user, the Flask web application, the database, the token serializer and the mail server interact with each other. This works, as first, the user requests the reset form via the `/reset_password` endpoint, then provides a new password (via the `/set_new_password` endpoint). Through a response, the server attaches the form corresponding to the reset request where the user can key in his registered email address.

When it is submitted the Flask application will check the database through SQLAlchemy and will find out whether the given email is in use or not. In case of a matching user account, time sensitive token is generated with the help of the `URLSafeTimedSerializer` and the reset email is created and sent to the user via the Flask-Mail service. In case no user is traced, the system continues to render a confirmation template without revealing that there was no such account, and none of sensitive data could be issued to possible criminals.

When the user gets the reset mail and visit the link, this time a GET request is made by the browser to `/reset_password/<token>`. Flask application then tries to check the token against some predetermined expiration (e.g. 30 minutes). In case the token is valid, the reset form will be rendered, otherwise, the user will be redirected with an error message being flashed.

When the user has entered a new password in the form and has sent the form, the application will again access the database, retrieve the user instance and update the password field to be a new hash value and commit the change. After that, the user is directed back to the login page and a success message will be received notifying the user that his/her password was updated successfully.

This diagram reflects the token-based safe mechanism adopted in the contemporary web applications to enable password renewal without violating user privacy and admitting unauthorized entries. It highlights such best practices as detection of non-disclosure of account existence and time-limited, cryptographically secure tokens.

### 3.3.4 Activity diagram:

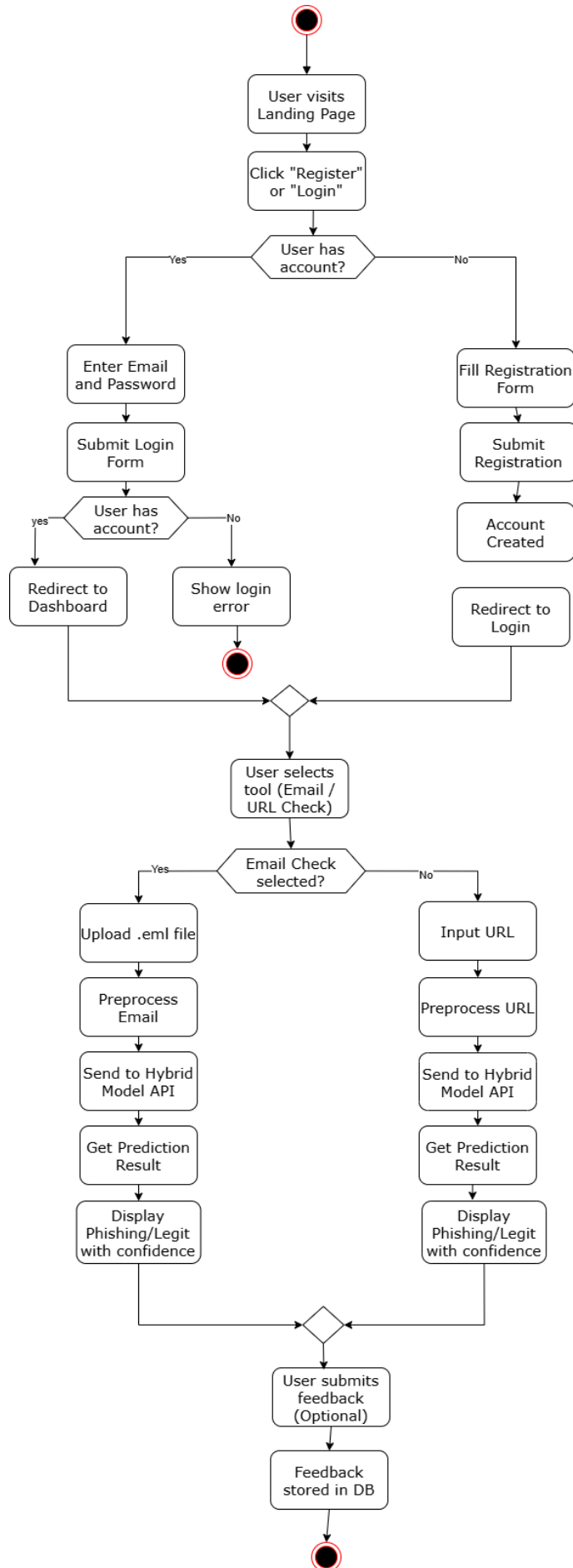


Figure 1.6: Activity diagram

The Activity diagram given in Figure 1.6 depicts the entire process of the user experience in using the PhishGuard web application starting with opening the landing page, ending with submitting feedback about the phishing activity. The proposed flow diagram shows the chronological process and decision making stages through which the users are directed into analyzing threats based on the input of either an email or a URL-based account. It provides an ideal representation of the application functioning logic and usability.

This starts by a user visiting the homepage and deciding to either log in or to register. When the user lacks an account, he or she is asked to complete the process of registration. Once they submit it successfully, their account is created and they are redirected to log in page. In case the user already has an registered account, he/she goes through the process of filling in his/her email and password and entering the sign in form. The system checks such credentials and in case of verification redirects the user to his dashboard. In case the log-in process is unsuccessful, the error message is put up.

After the authentication process, the user gets to choose among two existing phishing detection tools: Email Check or URL Check. In case of Email Check, the user uploads a file with an extension.eml to it, which is handed over to the hybrid machine learning model through the backend API and preprocessed. The model checks the content and renders a judgment giving the email as phishing or not and a degree of confidence. Alternatively, in case the user decides to use the URL Check, a suspicious web address is entered, which is preprocessed, as well as forwarded to the hybrid model. The result is then shown by comparable classification and confidence levels.

After the detection of the phishing result, the users can be invited to provide feedback on the accuracy of that prediction. When feedback is made, it will be kept under the database of the system so that in future the model correction can be made as well as the monitoring would help.

This is how the process then ends by providing the user with secure authentication, phishing analysis and community-based learning due to feedback systems.

### 3.4 System Requirements

#### 3.4.1 Requirement Identification

The purpose of this system is to detect phishing attempts in emails and URLs through a hybrid deep learning and machine learning model. It allows users to submit suspicious content via a web interface and receive real-time predictions, while also enabling admin monitoring and model management.

#### 3.4.2 Functional Requirements

Table 1.2: Functional Requirements

ID	Functional Requirement
FR1	The system must allow users to <b>register</b> and <b>log in</b> securely.
FR2	Users must be able to <b>submit an email (header + body)</b> for phishing detection.
FR3	Users must be able to <b>submit a URL</b> for phishing detection.
FR4	The system must display the <b>detection result</b> using the hybrid model.
FR5	Users must be able to <b>submit feedback</b> on prediction results.
FR6	The admin must be able to <b>log in</b> securely.



FR7	The admin must have access to a <b>dashboard</b> displaying user logs, prediction history, and feedback.
FR8	The admin must be able to <b>view and manage user messages</b> submitted through the contact form.
FR9	The admin must have the ability to <b>upload/retrain the detection models</b> .
FR10	The system must store all relevant data including user info, scan results, and feedback using <b>SQLite via SQLAlchemy</b> .

### 3.4.3 Non-Functional Requirements

Table 1.3: Non-Functional Requirements

ID	Non-Functional Requirement
NFR1	The system must return prediction results in <b>under 2 seconds</b> for each scan.
NFR2	The frontend must be <b>responsive</b> across devices (desktop, tablet, mobile).
NFR3	All user data must be <b>stored securely</b> with proper form validation and session management.
NFR4	The application should be <b>easy to maintain and extend</b> , following <b>three-layer architecture</b> .
NFR5	The system should support <b>up to 50 concurrent users</b> without performance degradation.
NFR6	The backend must be built with <b>Python and Flask</b> , using <b>modular code structure</b> .
NFR7	Email and URL scan results must be logged for <b>audit and improvement</b> .

### 3.5 Hardware:

- A system with Intel i5/i7/i9 with 10th Gen or later, or AMD R5/R7/R9 with 5000 series or later, fast storage and at least 8 GB RAM.
- As to the training of the rule-based deep learning models, but most notably, those using much computational power, it is suggested to use Nvidia RTX 2080 TI or use the more powerful GPU.ning the rule-based deep learning models, especially those requiring significant computational resources, an Nvidia RTX 2080 Ti or a more modern GPU is recommended. Alternatively, GPU support can be available from Kaggle or Google Colab for free to use platforms as well. Python is used commonly for model development.
- Hugging Face Transformers to prepare and adjust the models Moreover, PyTorch to train and apply the rule-based deep learning model.
- Relational databases such as SQL or non-relational databases, known as NoSQL, for archiving the student performance records and course prerequisites.
- Kaggle or Google Colab for training and testing or experiments of the model.r AMD R5/R7/R9 (5000 series or later) processors, featuring fast storage and at least 8 GB of RAM.
- For training the rule-based deep learning models, especially those requiring significant computational resources, an Nvidia RTX 2080 Ti or a more modern GPU is recommended. Alternatively, GPU support can be accessed through platforms like Kaggle or Google Colab, which offer free plans.

### 3.6 Software:

- Python for model development.
- Machine Learning Frameworks like Hugging Face Transformers for model development and fine-tuning and PyTorch For training and implementing the rule-based deep learning model.
- SQL or NoSQL databases for managing and storing student performance data and course prerequisite information.
- GitHub for version control.
- Kaggle or Google Colab for model training and experimentation.

## CHAPTER FOUR

### RESULT AND DISCUSSION

#### 4.1 Purpose of the System

The goal of the PhishGuard system is to create an intelligent and real-time phishing detection system and analyze it with the help of Natural Language Processing (NLP) and Deep Learning (DL). Enhancing the security of users will be aimed at detecting the potential malicious content and classifying it to ensure that users will be able to monitor phishing threats effectively using a secure web interface.

#### 4.2 System Architecture and Features

The system will be developed as modular so as to be maintainable, scalable and user friendly. The main functionalities, provided in the PhishGuard system are:

- **Registration & Authentication:**

The tokenized links enable secure log in, log out, as well as reset of passwords through email.

- **Email Scanner Module:** This module will scan the email files with the extension of .eml to recognize the possibility of phishing. When uploaded, the system pulls data features associated with the body of the email, email header, email domain, and links. The texts

are cleaned up, and tokenized and transformed into a padded sequence, as well as TF-IDF vectors (word- and character-level) are created and metadata including link count and domain encodings. The features are then passed into the hybrid deep learning model that integrates the advantages of using Convolutional Neural Networks (CNN) in extracting local features and Bidirectional Long Short-Term Memory (BiLSTM) networks in understanding the context sequence. The model produces a phishing probability score with which the emails can be marked as phishing or valid with high accuracy.

### **Module URL Scanner**

A URL Scanner is placed, which is tasked to evaluate relevancy of an address as to maliciousness. Users are allowed to simply enter any raw URLs which will then be converted into a rich feature vector comprising of TF-IDF-encoded representations of n-grams as well as 87 hand-designed numeric features representing structural information about the URL length, number of digits or symbols present, and other special patterns. The features are downsampled and summed along a vector input, and a deep learning model is used to interpret the input; this can be a shallow dense network, a deeper batch-normalization model, or a meta-ensemble classifier that has been trained on two-labeled data. The model comes up with the phishing probability score and one can detect phishing attempts accurately using the characteristics of URLs.

- **Prediction Logging**

There is metadata such as user, confidence score, and timestamp of all scans (email or URL).

- **User Dashboard**

The registered users can see pie and line graphs of their scanning history and receive world statistics about phishing detection.

- **Admin Ergo Viewer**

Admin-only page to read contact messages (public) made with the contact form.

- **APIs by Rest Endpoints**

Email, URL, User specific history, and global system stats based on JSON APIs.

## **4.2 User Interface Modules**

The PhishGuard system is clean, responsive, and easy to navigate with obvious divisions between the functions expected to be used by general users and the functions used in administration by the administrators. Each of the modules works towards providing a seamless experience and covers phishing detection in URL vectors, email vectors, and allows safe user interactions and the administration-side monitoring.

### **4.2.1 Login Page**

phishguards.com The home screen is the log-in page of the PhishGuard. It consists of the email and password input fields where their correctness is checked to avoid incorrect data entry. There is an optional box to remember the user session at the log in page. There is also a link to users who do not have an account where they are referred to the module for making registrations. After a semblance of success in its authentication process, a user session is triggered and the user is redirected to the main dashboard. Such an element is realized with the help of Flask and WTForms and designed with Bootstrap 5. To prevent security concerns, all the passwords are hashed then stored in the SQLite database through the SQLAlchemy code, providing the best practices on user authentication.

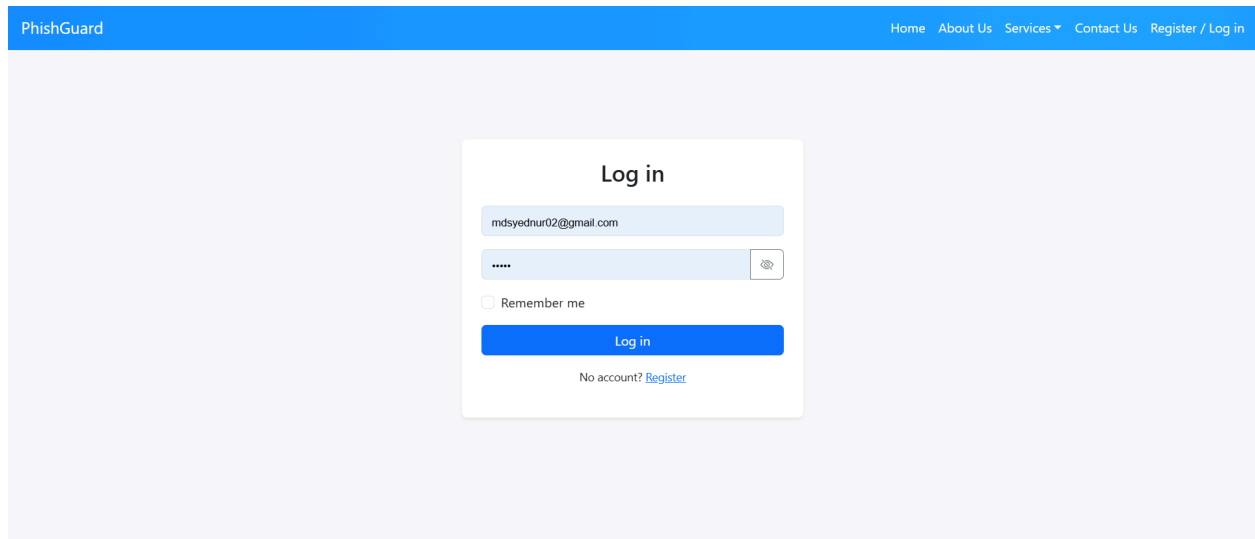


Figure 1.7: log-in page

#### 4.2.2 Registration Page

Registration module facilitates the creation of an account by new users within the system. It has a field to enter the email, password, and the confirmation field of the password with real-time validation so that there will be consistency and preventing the creation of a duplicate entry. When the form has been successfully submitted, the user credentials are encrypted with the help of the SQLAlchemy SQLite with Object-Relational Mapping (ORM) and stored in the User table of the SQLite database. It is a module simplifying the process of account creation and retaining the levels of security and usability by following the principles of intuitive design and responsiveness in feedback.

Log in'."/>

PhishGuard

Home About Us Services Contact Us Register / Log in

### Create account

Email

Password

Confirm password

Register

Already registered? [Log in](#)

Figure 1.8: Registration page

#### 4.2.3 Home Page (Landing Page)

The home page is the outward landing part of the PhishGuard where the mission of the platform is supposed to be presented, in this case, it is real-time detection of phishing in emails, URLs. It also involves a captacious heading, Stop Phishing in Its Tracks, with a call to action button, Get Started. The most important platform stats can be seen clearly beneath, i.e. the total emails scanned, the number of malicious URLs marked and the overall accuity. This landing page is deliberately lightweight and speedy loading and is optimized through HTML CSS and Bootstrap to be able to access all gadgets.

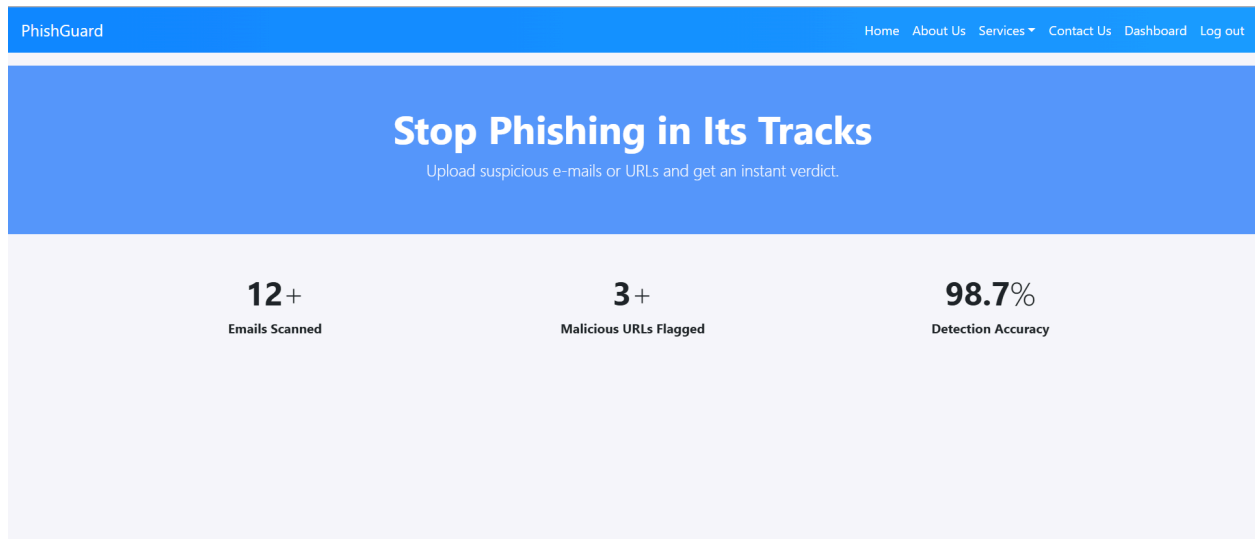


Figure 1.9: Home page

#### 4.2.4 Dashboard

After successful log in, the users will be redirected to the dashboard, which is the central control console of the system. In this case, Internet users get two main choices including the Email Scanner and the URL Scanner. They are shown in the form of interactive cards to be easily tapped. In the case of administrators, there is an inbox management module in the dashboard. The dashboard has been dynamic, and it depends on the user role so that the sensitive administrative updates are not visible to ordinary users. Such role-based interface segregation is applied on the backend layer as well as frontend layer.



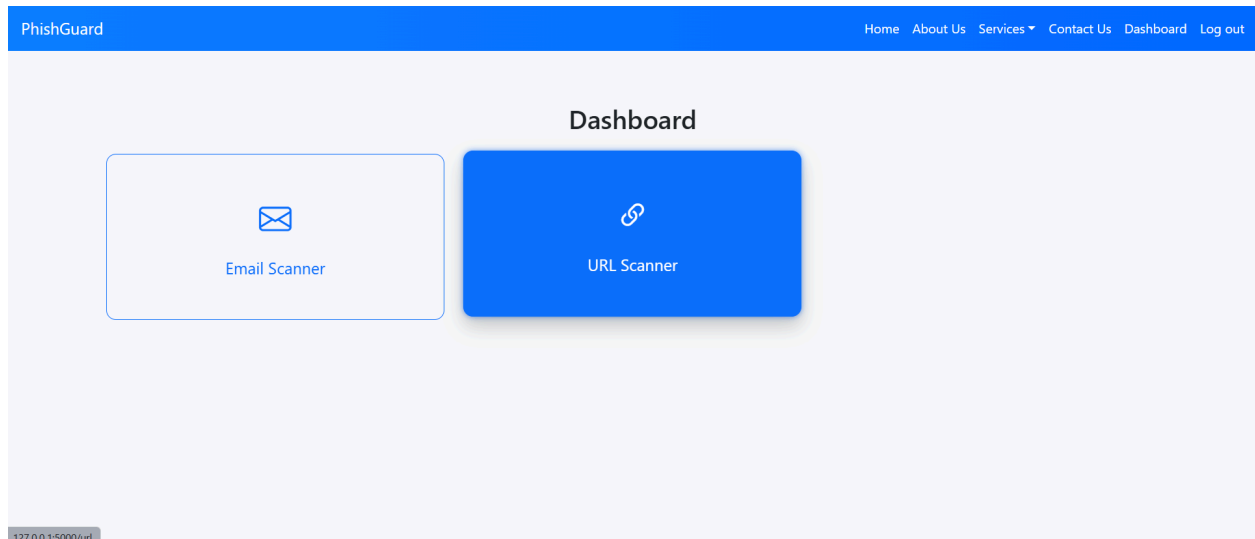


Figure 1.10: Dashboard

#### 4.2.5 Email Check Module

Email Phishing Detection module offers a simplified interface to study .eml files of emails. A user is able to upload email file and scan using the scan button. The uploaded file is then processed using a hybrid deep learning model, which is triggered with the help of the `get_email_predict()` function placed in the `utils/email_model.py` module. The user is presented with the result which is in the form of a donut chart where the user can see the result of the classification (e.g. Legitimate or phishing) and the percentage confidence corresponding to it (e.g. 99.0 % Legitimate or 60.7 % Phishing). It uses simplicity in user interaction with strong predictive analytics in this module.

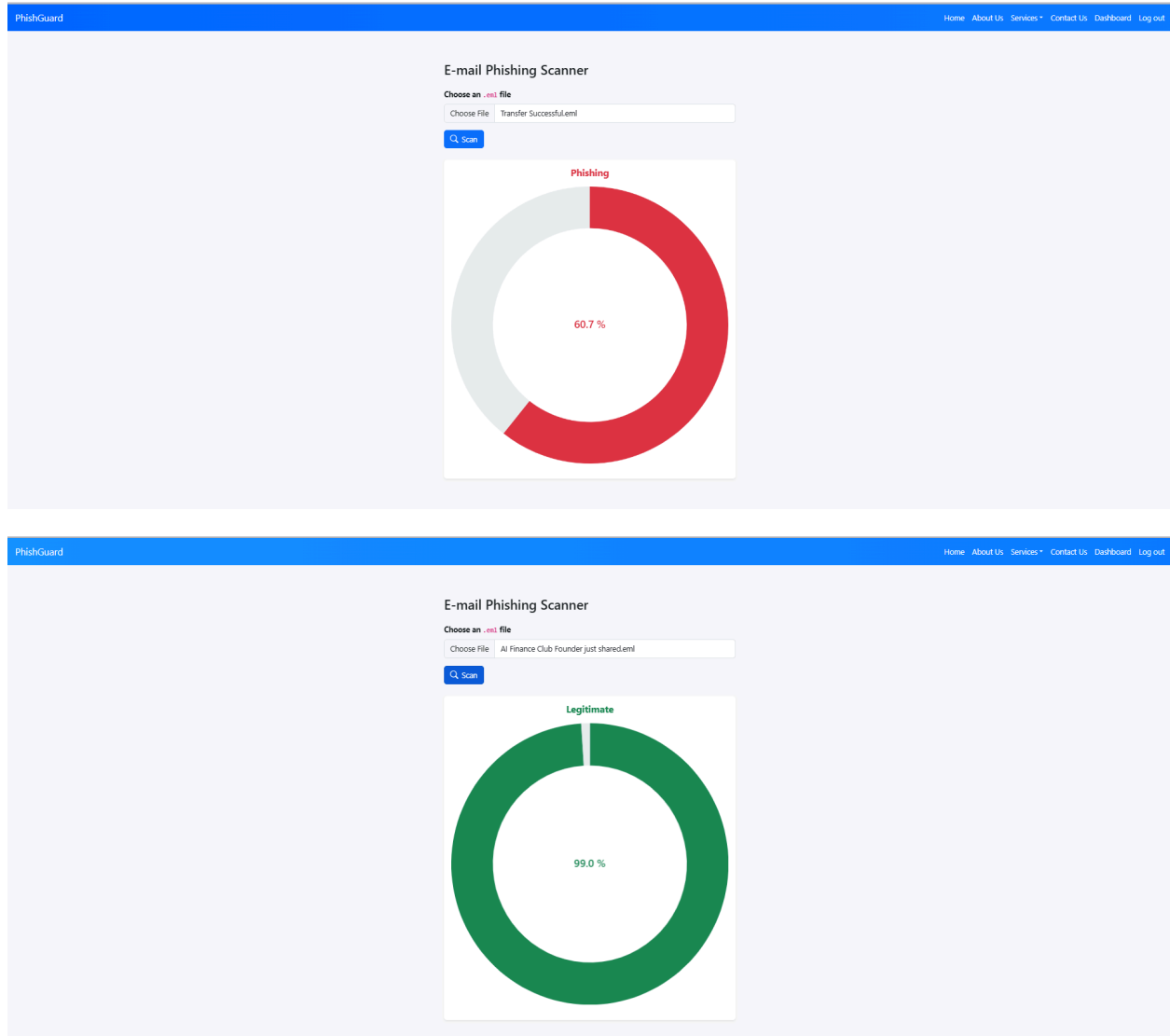


Figure 1.11 and 1.12: Email Check Module

#### 4.2.6 URL Check Module

URL Phishing Detection module is a tool that enables the input of any URL, which is placed in a text field and is analyzed using the phishing attributes. When you click on scan, the system will process the input using the `get_url_predict ()` method that will call the trained hybrid model used in the file `utils/url_model.py`. The output contains a classification response, either "Phishing" or "Legitimate", and an associated score (e.g. 95.6% Legitimate). Such a module is used in the system as facilitating the real-time inference (with a low amount of latency) and serves as a critical component of applying this system to detect the threats via a web-based format.

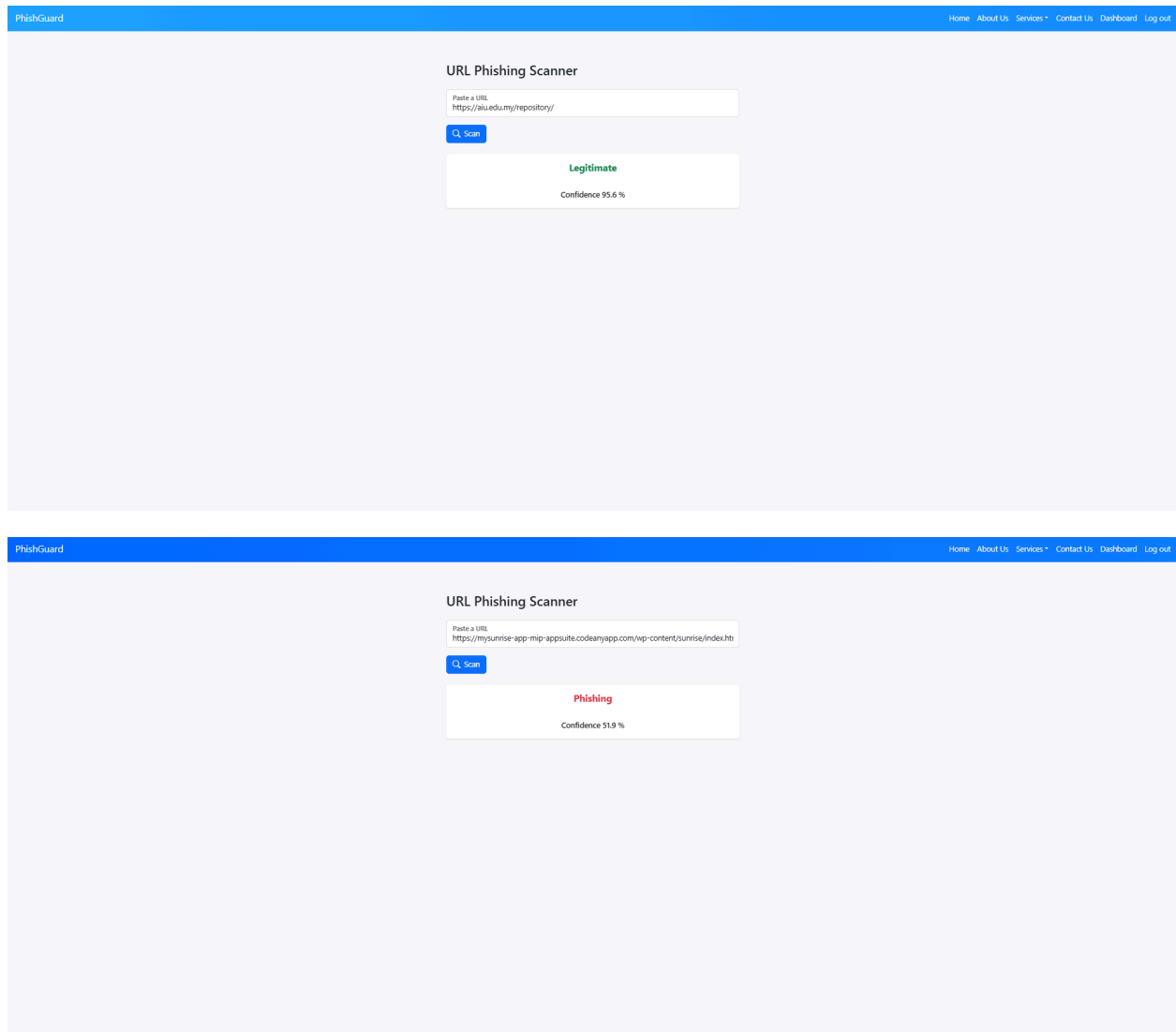


Figure 1.13 and 1.14: URLCheck Module

#### 4.2.7 Admin Panel

The administrative users get to see more features not accessible to ordinary users. These are the Inbox Management Module whereby the feedback messages sent using the contact form appear

in a tabular format with the name, email, timestamp and the body of the message. This allows administrators to track user issues and determine rendering problems or features. Moreover, there will be a futuristic part, the Model Monitoring and Management Module, that will be used to retrain or update the detection models through control at the backend. Secure session handling and role-based route protection are applied to all the administrator modules to make sure that important information and controls can be accessed by the authorized users only.

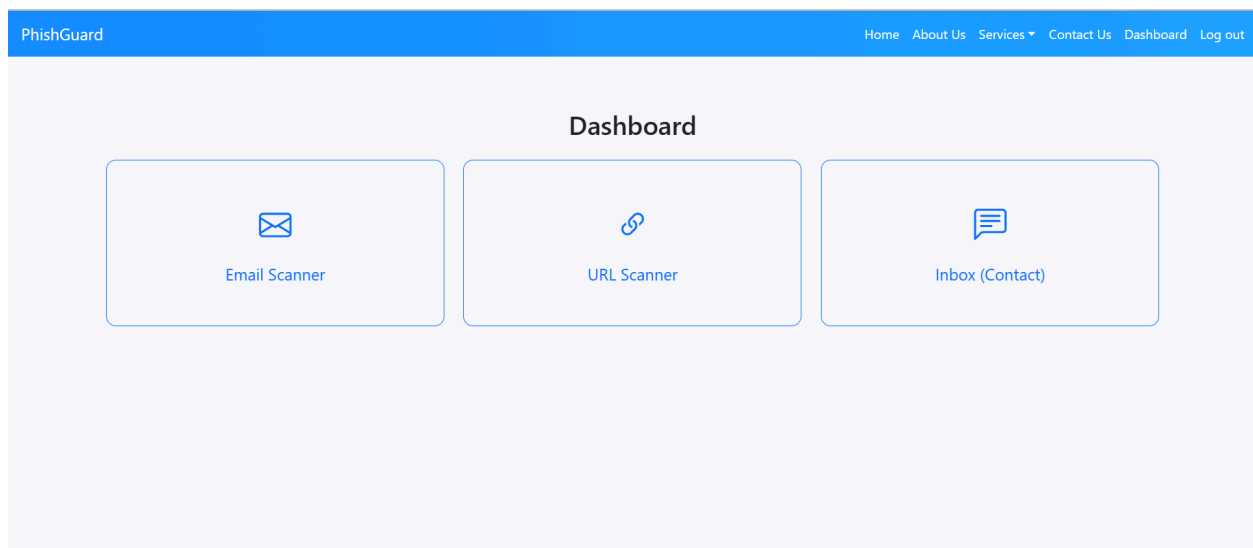


Figure 1.15: Admin Panel

#### 4.2.4 System Integration

PhishGuard phishing detection system combines various system elements, including user interface, application logic, and database into a unitary and responsive structure. Such integration lets perform phishing detection in real-time, secure data processing as well as fluent interaction among modules, but maintains modularity and scalability. The general structure of integration provides that the interactions demonstrated by the user be converted directly into a backend activity and remain in structured format, available to audit, feedback, or inferential purposes.

#### **4.2.4.1 API Gateway: Frontend to Backend Communication**

The feature of the integration strategy in the system is the following concept and utilisation of Flask routing mechanisms as an internal API gateway. Communication between the frontend (created with HTML, Bootstrap, and Java Script with AJAX) and backend business logic is made possible with the use of these routes. The inputs provided by the users when using the interface elements (e.g. the Email Scanner, or the URL Scanner), are posted via regular forms or asynchronous AJAX interactions. The backend Flask takes care of the respective routes:

- The ways of authentication and user management: /login and /register
- /check\_email to submit files in the .eml format and predict phishing
- url phishing checking by textual URL entry /check\_url
- /reset\_request and /reset\_token in order to have a safe password reset mechanism with the help of a token validation method

The AJAX is mostly used in the scanner components to improve the user experience as it gives the scans synchronous results which is real time without full page refresh. The backend sends back responses in JSON format that each response will go under phishing categories (e.g., Phishing or Legitimate) and the level of confidence. This architecture contributes to a faster response time, lower latency, better testing and debugging of each module.

#### **4.2.4.2 Database Integration: User Management, Predictions, and Feedback Logs**

PhishGuard has an SQLite database as the backend in combination with the application using SQLAlchemy ORM. All long-term information, such as user records, system predictions, user feedback, is stored in and retrieved by using model-based abstractions that obviate use of raw SQL. Data storage is based on three major types of database models:

User Model:Manages user name/secret (hashed with triple DES), email ID and role flag (administrative).

Prediction Model: Records every instance where phishing is detected including user ID, type of input (should be email or URL), the prediction, and the confidence score, and time.

ContactMessage Model: Stores and retrieves the messages left via the Contact Us form so that the administration can follow-up the messages and evaluate the system.

The data transactions are performed with `db.session.add()` and after this, the information is facilitated with `db.session.commit()` to make the information durable. Automatic rollback features ensure that in case of transaction failure, only the complete test of the result is entered, preventing half baked or dirty entries into the database. Moreover, the access to the database will also go through the application logic layer exclusively, and the frontend does not directly communicate with it. This isolation facilitates security, maintainability, as well as, long term scalability of the platform.

#### **4.3 Model Development and Workflow**

The PhishGuard system is based on one hybrid deep learning method to end the phishing attempts on the email content as well as the URLs. Model development process includes the elements of natural language processing (NLP), deep neural networks, and ensemble logic to

produce consistent real-time results. There is also the role of a feedback loop mechanism into the workflow which allows progressive model improvement through user feedback and performance records.

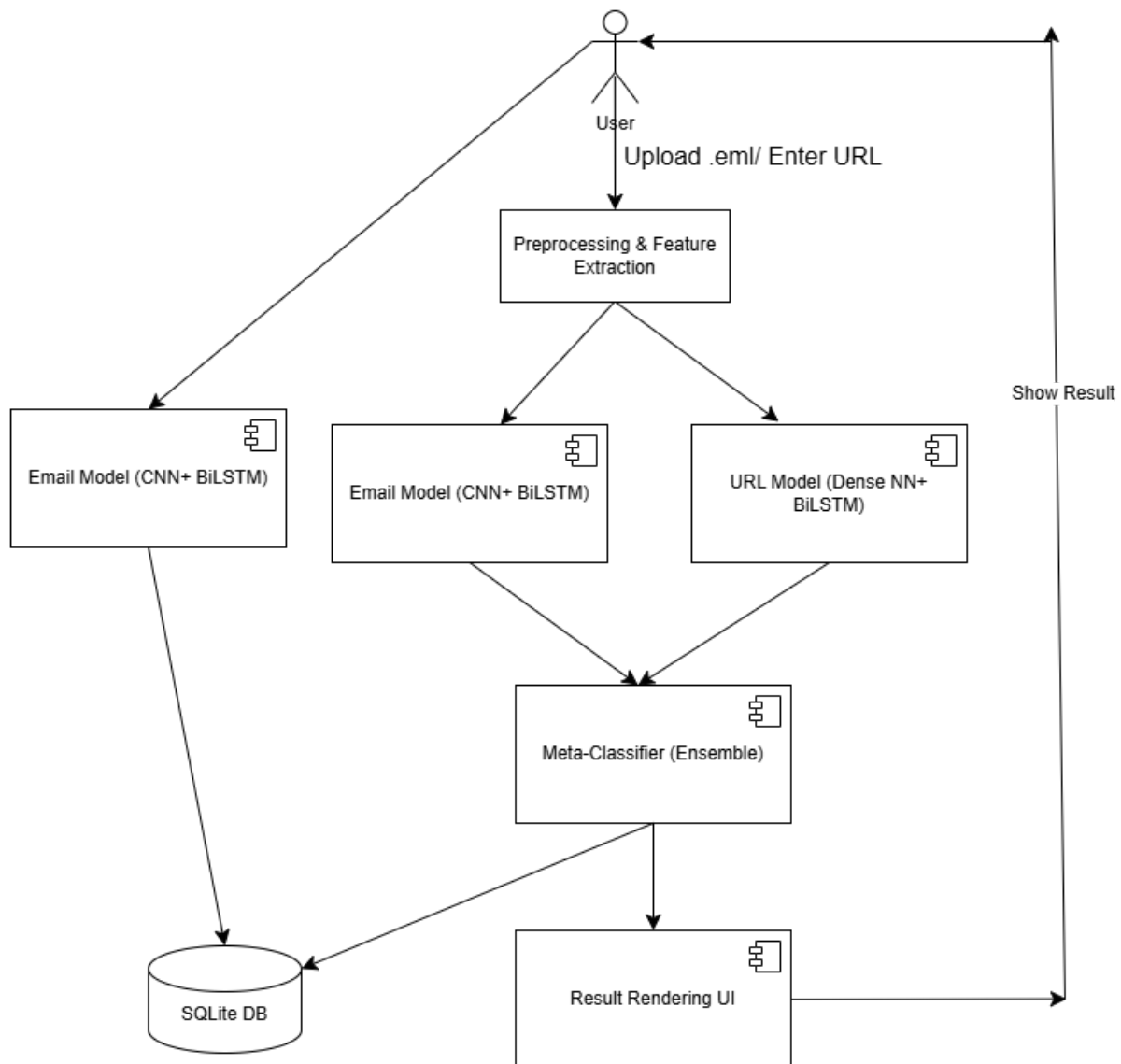


Figure 1.16: Model Development and Workflow

### 4.3.1 Hybrid Model Architecture

That phishing detection framework uses two differently trained deep learning models on email and URLs as input, which are merged in an ensemble meta-classifier:

Email Model:

- Input: .eml files (email header + body)
- Characteristics: text and metadata preprocessed tokens
- Architecture: There is a sequence of both Bidirectional LSTM (BiLSTM) and CNN layers to both model sequential dependencies and some local patterns.

URL Model:

- Input Raw URL strings
- Features: These features were obtained based on TF-IDF, character-level encoding, domain vectorization, and the URL structure patterns.
- Architecture: Combined feature vectors used in dense neural network or Logistic Regression meta-model
- Outputs of each of the classifiers are combined with a meta-classifier layer, which delivers a final decision and a degree of confidence.

#### 4.3.2 Data Flow Pipeline

The system follows a structured data flow:

1. **User Input:** Via frontend (upload .eml or enter URL)
2. **Preprocessing:** Text cleaning, vectorization (TF-IDF, Tokenizer), encoding
3. **Prediction:**
  - Email: `get_email_predict()` → Hybrid DL model → JSON response



- URL: `get_url_predict()` → Feature fusion → Prediction via ensemble

4. **Result Display:** Rendered as visual feedback (donut chart or textual verdict)

5. **Logging:** Input type, result, and timestamp stored in SQLite via SQLAlchemy

This pipeline ensures modularity, maintainability, and easy debugging for both development and deployment phases.

### 4.3.3 Feedback Loop for Continuous Improvement

In order to make system refinements possible, the system encompasses a user feedback system:

- The records of Contact Form Submissions are kept in the `ContactMessage` table.
- Admins can access feedback messages via the inbox module (e.g., false positives or suggestions of improvements).
- Predictions and user comments encompass a ground truth data set that can be utilized to possibly retrain and fine-tune models.
- This loop forms a basis to semi-supervised learning and performance of the application in the real world made on the basis of interactions that users actually have.

## 4.4 Testing

### 4.4.1 Unit Testing

In the PhishGuard phishing detection system in the system was unit tested where the accuracy of each component part was established and also in Python the modules that are run in the backend or backend modules where the parts were tested to be accurate and reliable. The most basic functions like `get_email_predict()` and `get_url_predict()` were thoroughly tested in order to get accurate forecasts, rightful exception tracking and a regularity of response structure returned. The authentication process of the system was also checked at the level of the route, such as `/login`

and /register, valid input validation, user feedback, and integration with the database by means of SQLAlchemy. The security and expiry processing involving token generation and verification of password reset functionality were validated with the help of the dangerous module. Also, scan endpoints of AJAX were unit checked to ensure that they properly took user input and responded in JSON in real-time. These unit tests were crucial towards testing the modules individually so that few errors are encountered during testing and deployment of the system across the board.

Table 1.4: Unit Testing

Component	Functionality Tested	Test Scenario	Expected Result
get_email_predict()	Email phishing detection via DL model	Upload a valid .eml file	Returns prediction label: "Spam" or "Not Spam"
get_url_predict()	URL phishing detection using hybrid model	Submit a suspicious URL	Returns prediction: "Phishing" or "Legitimate"
generate_token()	Password reset token generation	Generate token for a valid user email	Returns a secure time-sensitive token
verify_token()	Token validation	Submit a valid/expired token	Valid token returns user object; expired returns None

/login route	User authentication	Submit correct/incorrect credentials	Redirects to dashboard on success, error message on failure
/register route	New user registration	Submit valid/duplicate user data	New user added or shown “user already exists” message
/scan/email	Email scan API logic (AJAX)	AJAX call with uploaded file	JSON response with prediction label and score
/scan/url	URL scan API logic (AJAX)	AJAX call with user-input URL	JSON response with prediction label and score
ContactMessage	Contact form processing	Submit contact form with all fields filled	Message stored in database, success flash message shown
SQLAlchemy ORM	User and Prediction model interactions	Add, retrieve, and delete records	Database reflects correct insertion/deletion

#### 4.4.2 System Testing

System testing was performed to reify the workability and compatibility of all the modules of the phishing detection system. This testing supported the fact that every component of the system including the login, registration, email and URL phishing detection, feedback, and the administration existed within a unified environment in their real world usage scenario. As an example, the modules dealing with logging in and registration data were first checked whether they had appropriate authentication and validation schemes and the phish-detecting modules were tested and proven to yield accurate data depending on the data entered by the user. An assessment was done on the dashboard to ascertain an easy interface to retrieve information and navigate through and on the administrator panel to verify whether it was primitive to manage users, predictions, feedback, and the model until such without difficulties. Testing was done on each module whereby it was tested against the anticipated results to determine the reliability, user-friendliness as well as how it reacted to varying situations.

Table 1.5: System Testing

<b>Component/Module</b>	<b>Test Scenario</b>	<b>Expected Result</b>
Login Module	Valid and invalid user login attempts	Successful login for valid users, error message for invalid ones

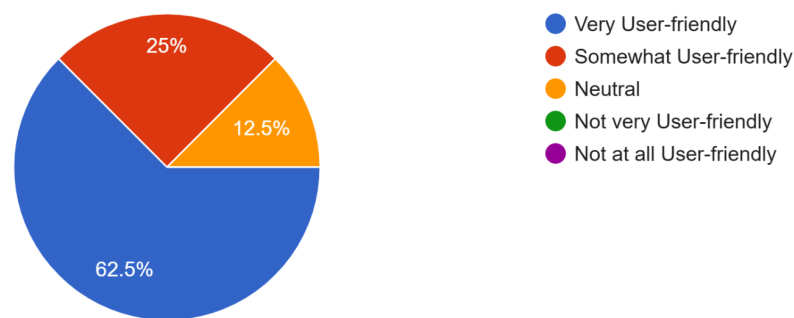
Registration Module	Submit form with complete and incomplete data	Account created if valid, validation errors shown otherwise
Email Phishing Detection	Submit raw email text for analysis	Phishing or not-phishing label returned accurately
URL Phishing Detection	Input URL to check phishing status	URL classification as phishing or safe
Dashboard	Navigate to dashboard post-login	User-specific data and prediction logs displayed
Feedback Submission	Submit feedback form	Feedback stored and success message shown
Admin Login	Admin enters correct and incorrect credentials	Admin access granted or denied accordingly
Admin Dashboard	View all user logs, predictions, and feedback	Data retrieved and displayed correctly

Model Management	Upload/update detection models	Model updated and system ready for prediction
Contact Message Management	Admin views and deletes contact messages	Messages loaded/deleted as requested

#### 4.4.3 User Testing

It conducted user testing in order to test the usability of the phishing detection system on an end-user level and how it operates. The platform was used by real users: students and ordinary people were used to complete basic functions registration, log in, email and URL uploading in order to analyze phishing attacks and feedback submission. The goal was to have a system that is intuitive, responsive and with no usability bottlenecks.

How user-friendly did you find the PhishGuard interface?  
8 responses



Rate your overall satisfaction with PhishGuard:  
8 responses

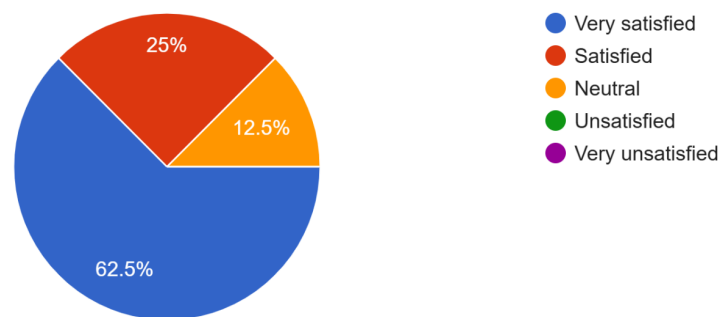


Figure 1.17 and 1.18 : User Feedback

Table 1.6: User Testing

Scenario	User Action	Expected Outcome
----------	-------------	------------------

Registration	User signs up with email, password, and role	Account is created and confirmation message displayed
Login	User logs in with valid credentials	Redirected to dashboard with access to detection tools
Email Phishing Detection	User submits email content	Prediction result displayed instantly without page reload
URL Phishing Detection	User inputs suspicious URL	System classifies URL and displays phishing probability
Feedback Submission	User submits feedback from result page	Feedback saved and user notified of successful submission
Admin Login	Admin logs in with correct credentials	Redirected to admin dashboard
Admin View Prediction Logs	Admin checks user history and results	Logs are correctly fetched and displayed



Admin Manage Messages	Admin views and responds to user contact messages	Messages are accessible and manageable
-----------------------------	---	---

## CHAPTER FIVE

### CONCLUSION AND FUTURE WORK

#### Conclusion

This study was able to design and build an intelligent system that can detect phishing attacks i.e., PhishGuard, which uses the effectiveness of Natural Language Processing (NLP) and Deep Learning (DL) to detect the presence of phishing attacks both on the mails and links. The system eliminates the shortcomings of the conventional heuristic-based models through adoption of the complex text analysis, extraction of features and the hybrid classification leading to better accuracy and adaptability in addition to reducing the false positive rates.

Being developed based on the Incremental Development Model, PhishGuard has a modular structure, and the user interface is based on Flask, being responsive and secure, and the APIs related to the predictions work in real time. The fact that extensive unit, system, and user testing

have been conducted proves that the system works reliably in the real world environment. The model yields quite strong results in various measures of evaluation, like precision, recall, and confidence scores.

Other than its technical significance, the system is helpful in relation to SDG 16: Peace, Justice, and Strong Institutions by shifting the likelihood of cybercrime and recognizing and strengthening confidence in online communication channels, and stimulating equally safe engagement in online actions and activities. PhishGuard enables users and organizations to have optimal cybersecurity defense tools that achieve safer, less opaque digital spaces crucial to governing ethically and instilling resilience on institutions.

## **Future work**

In order to optimize the success of PhishGuard and maximize on scalability, the future improvements which are proposed to be undertaken include the following:

- **Multilingual Phishing Detecting:**  
Increase the system ability to detect phishing in different languages to allow supporting users all over the world and preventing attacks beyond English-based content.
- **Real-Time Threat Feeds Integration:**  
Include real-time Threat intelligence (e.g. threat databases, dark web) to dynamically refresh detection pattern and allow flexibility to new phishing methods.
- **Portable and low weight-deployment:**  
Execute it on hard and resource-limited systems that are more viable to serve customers in low-infrastructure areas and enhance access among users.
- **Self-Learning, Learning Online:**  
Institute online training processes in which the model can constantly be retrained and refined according to new feedback and the results of new detections, in order to enable adaptive continuous evolution of the model.
- **Integration of Browsers and Email Clients:**  
Write browser plug-ins or email clients extensions to be directly integrated so that the emails and links will be scanned in real-time before the user gets to interact with them.
- **Integration of explainable artificial intelligence (XAI):**

Improve clarity in models, to help users and administrators to know why a given email or URL was labeled phishing due to understanding the explainability mechanism used (e.g., LIME, SHAP).

In developing these directions, the PhishGuard may develop into a more inclusive, competent and people-friendly cybersecurity tool and expand its influence on the digital security, institutional credibility and can contribute to sustainable growth of a digital world.

## 6. References :

- Alhogail, A., & Alsabih, A. (2021). Applying machine learning and natural language processing to detect phishing email. *Computers & Security*, 110, 102414.
- Aljofey, A., Jiang, Q., Qu, Q., Huang, M., & Niyigena, J. P. (2020). An effective phishing detection model based on character level convolutional neural network from URL. *Electronics*, 9(9), 1514
- Atawneh, S., & Aljehani, H. (2023). Phishing email detection model using deep learning. *Electronics*, 12(20), 4261..
- Butt, U. A., Amin, R., Aldabbas, H., Mohan, S., Alouffi, B., & Ahmadian, A. (2023). Cloud-based email phishing attack using machine and deep learning algorithm. *Complex & Intelligent Systems*, 9(3), 3043-3070
- Benavides-Astudillo, E., Fuertes, W., Sanchez-Gordon, S., Nuñez-Agurto, D., & Rodríguez-Galán, G. (2023). A phishing-attack-detection model using natural language processing and deep learning. *Applied Sciences*, 13(9), 5275.
- Çolhak, F., Ecevit, M. İ., Uçar, B. E., Creutzburg, R., & Dağ, H. (2024). Phishing Website Detection through Multi-Model Analysis of HTML Content. *arXiv preprint arXiv:2401.04820*.
- Cazares, M. F., Andrade, R., Navas, G., Fuertes, W., & Herrera, J. (2021, July). Characterizing phishing attacks using natural language processing. In *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)* (pp. 224-229). IEEE.
- Coyac-Torres, J. E., Sidorov, G., Aguirre-Anaya, E., & Hernández-Oregón, G. (2023). Cyberattack detection in social network messages based on convolutional neural

- networks and NLP techniques. *Machine Learning and Knowledge Extraction*, 5(3), 1132-1148.
- El Aassal, A., Baki, S., Das, A., & Verma, R. M. (2020). An in-depth benchmarking and evaluation of phishing detection research for security needs. *Ieee Access*, 8, 22170-22192.
- Elsadig, M., Ibrahim, A. O., Basheer, S., Alohal, M. A., Alshunaifi, S., Alqahtani, H., ... & Nagmeldin, W. (2022). Intelligent deep machine learning cyber phishing url detection based on bert features extraction. *Electronics*, 11(22), 3647.
- Elsadig, M., Ibrahim, A. O., Basheer, S., Alohal, M. A., Alshunaifi, S., Alqahtani, H., ... & Nagmeldin, W. (2022). Intelligent deep machine learning cyber phishing url detection based on bert features extraction. *Electronics*, 11(22), 3647.
- Gutiérrez, L. F., Abri, F., Armstrong, M., Namin, A. S., & Jones, K. S. (2020, December). Email embeddings for phishing detection. In *2020 IEEE International Conference on Big Data (Big Data)* (pp. 2087-2092). IEEE.
- Halgaš, L., Agrafiotis, I., & Nurse, J. R. (2020). Catching the Phish: Detecting phishing attacks using recurrent neural networks (RNNs). In *Information Security Applications: 20th International Conference, WISA 2019, Jeju Island, South Korea, August 21–24, 2019, Revised Selected Papers 20* (pp. 219-233). Springer International Publishing.
- Haynes, K., Shirazi, H., & Ray, I. (2021). Lightweight URL-based phishing detection using natural language processing transformers for mobile devices. *Procedia Computer Science*, 191, 127-134.
- Heiding, F., Schneier, B., Vishwanath, A., Bernstein, J., & Park, P. S. (2023). Devising and detecting phishing: Large language models vs. smaller human models. *arXiv preprint arXiv:2308.12287*.
- Kumar, A., Chatterjee, J. M., & Díaz, V. G. (2020). A novel hybrid approach of SVM combined with NLP and probabilistic neural network for email phishing. *International Journal of Electrical and Computer Engineering*, 10(1), 486.
- Koide, T., Fukushi, N., Nakano, H., & Chiba, D. (2024). ChatSpamDetector: Leveraging large language models for effective phishing email detection. *arXiv preprint arXiv:2402.18093*.
- Korkmaz, M., Sahingoz, O. K., & Diri, B. (2020, July). Detection of phishing websites by using machine learning-based URL analysis. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)* (pp. 1-7). IEEE.
- Li, Y., Huang, C., Deng, S., Lock, M. L., Cao, T., Oo, N., ... & Lim, H. W. (2024). KnowPhish: Large Language Models Meet Multimodal Knowledge Graphs for Enhancing Reference-Based Phishing Detection. *arXiv preprint arXiv:2403.02253*.

- Maneriker, P., Stokes, J. W., Lazo, E. G., Carutasu, D., Tajaddodianfar, F., & Gururajan, A. (2021, November). Urltran: Improving phishing url detection using transformers. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)* (pp. 197-204). IEEE.
- Mahendru, S., & Pandit, T. (2024). SecureNet: A Comparative Study of DeBERTa and Large Language Models for Phishing Detection. *arXiv preprint arXiv:2406.06663*.
- Milner, H., & Baron, M. (2024). Establishing an optimal online phishing detection method: Evaluating topological NLP transformers on text message data. *Journal of Data Science and Intelligence Systems*, 2(1), 173-181.
- Mittal, A., Engels, D. D., Kommanapalli, H., Sivaraman, R., & Chowdhury, T. (2022). Phishing detection using natural language processing and machine learning. *SMU Data Science Review*, 6(2), 14.
- Ozcan, A., Catal, C., Donmez, E., & Senturk, B. (2023). A hybrid DNN–LSTM model for detecting phishing URLs. *Neural Computing and Applications*, 1-17.
- Peng, T., Harris, I., & Sawa, Y. (2018, January). Detecting phishing attacks using natural language processing and machine learning. In *2018 IEEE 12th international conference on semantic computing (icsc)* (pp. 300-301). IEEE.
- Rabbi, M. F., Champa, A. I., & Zibran, M. F. (2024). Phishy? Detecting Phishing Emails Using Machine Learning and Natural Language Processing. In *Software Engineering and Management: Theory and Application: Volume 16* (pp. 119-137). Cham: Springer Nature Switzerland.
- Salloum, S., Gaber, T., Vadera, S., & Shaalan, K. (2022). A systematic literature review on phishing email detection using natural language processing techniques. *IEEE Access*, 10, 65703-65727.
- Salloum, S., Gaber, T., Vadera, S., & Shaalan, K. (2021). Phishing email detection using natural language processing techniques: a literature survey. *Procedia Computer Science*, 189, 19-28.
- Shaukat, M. W., Amin, R., Muslam, M. M. A., Alshehri, A. H., & Xie, J. (2023). A hybrid approach for alluring ads phishing attack detection using machine learning. *Sensors*, 23(19), 8070.
- Shirazi, H., Muramudalige, S. R., Ray, I., & Jayasumana, A. P. (2020, November). Improved phishing detection algorithms using adversarial autoencoder synthesized data. In *2020 IEEE 45th conference on local computer networks (lcn)* (pp. 24-32). IEEE.
- Verma, P., Goyal, A., & Gigras, Y. (2020). Email phishing: Text classification using natural language processing. *Computer Science and Information Technologies*, 1(1), 1-12.

- Verma, R., & Shashidhar, N. (2015). Automatic phishing email detection based on natural language processing techniques. US Patent App. 14/015,524.
- Yang, P., Zhao, G., & Zeng, P. (2019). Phishing website detection based on multidimensional features driven by deep learning. *IEEE Access*, 7, 15196–15209. <https://doi.org/10.1109/ACCESS.2019.2894183>
- Zhang, L., & Zhang, P. (2020, October). PhishTrim: Fast and adaptive phishing detection based on deep representation learning. In *2020 IEEE International Conference on Web Services (ICWS)* (pp. 176-180). IEEE.

## 7. appendix

### Gantt Chart

PROJECT TITLE		Phishing Attack Detection Model Using Natural Language Processing					
SUPERVISOR		Assoc. Prof. Dr. Basheer Riskhan				TEAM MEMBERS	
PROJECT DURATION		9 Months				STATUS	
WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE	WEEK 1
							M T W R F
<b>1</b>	<b>Project Plan</b>						
1.1	Pre Research	Team	10/01/24	10/07/24	6	100%	
1.1.1	Project Research	Team	10/8/24	10/20/24	12	100%	
1.2	Logbook Setup	Team	10/21/24	10/24/24	3	30%	
1.3	Conduct research on existing systems	Team	10/25/24	11/15/24	20	100%	
1.4	Finalize project proposal	Team	11/16/24	12/20/24	34	100%	
<b>2</b>	<b>Research and Planning</b>						
2.1	Progress Report	Team	12/27/24	1/27/25	30	30%	
2.2	Conduct research on existing systems	Team	12/29/24	1/7/25	8	80%	
2.3	Progress Report	Team	1/8/25	1/27/25	19	100%	
2.4	Project Pitching Preparation	Team	1/2/25	1/9/25	7	100%	
<b>3</b>	<b>Model Development</b>						
3.1	System Documentation	Team	1/28/25	2/11/25	13	0%	
3.2	Phishing Dependency Mapping	Team	1/28/25	2/11/25	13	0%	
3.3	System Design	Team	1/28/25	2/27/25	29	0%	
3.4	Create Database and rule	Team	2/12/25	3/13/25	31	0%	
3.5	Finalizing System Features	Team	3/14/25	3/28/25	6	0%	

3.6	System building	Team	3/21/25	4/30/25	39				
<b>4</b>	<b>Optimization and Finalization</b>								
4.1	Pilot Testing	Team	5/1/25	5/31/25	30	0%			
4.2	Testing and Validation	Team	5/1/25	5/31/25	30	0%			
4.3	System Deployment	Team	6/1/25	6/14/25	13				
4.4	Final Review & Adjustments	Team	6/15/25	6/21/25	6	0%			
4.5	Finalizing Progress report	Team	6/22/25	6/27/25	5				
4.4	Presentation and Thesis Submission	Team	4/27/24	6/27/25	420	0%			

#### Project Reference

Turnitin :





## 9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.




### Filtered from the Report

- Bibliography
- Quoted Text

### Match Groups

-  **81** Not Cited or Quoted 9%  
Matches with neither in-text citation nor quotation marks
-  **3** Missing Quotations 0%  
Matches that are still very similar to source material
-  **0** Missing Citation 0%  
Matches that have quotation marks, but no in-text citation
-  **0** Cited and Quoted 0%  
Matches with in-text citation present, but no quotation marks

### Top Sources

- 5%  Internet sources
- 4%  Publications
- 6%  Submitted works (Student Papers)

### Integrity Flags

#### 0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

## Originality

My Files

My Files

Upload File

Files per Page:

10 ▾

Title	Author	Similarity	AI	Actions
final year report (1).pdf	Mutasim Billah	9% <a href="#">Downloaded</a>	11*% <a href="#">Downloading...</a>	<a href="#">Delete</a>

Page 1 of 1

Showing 1 to 1 files

Page 1 of 1



Source code:

```
app.py > ...
1  # app.py - PhishGuard back-end
2  #
3
4  from __future__ import annotations
5  import os, time, datetime, logging, functools
6  from typing import Callable
7  from dotenv import load_dotenv
8  from flask import (
9      Flask, render_template, request, redirect,
10     url_for, flash, jsonify, abort
11 )
12 from flask_sqlalchemy import SQLAlchemy
13 from flask_login import (
14     LoginManager, login_user, login_required,
15     logout_user, current_user, UserMixin
16 )
17 from werkzeug.security import generate_password_hash, check_password_hash
18 from itsdangerous import URLSafeTimedSerializer
19 from flask_mail import Mail, Message
20 from sqlalchemy import func
21
22 # ----- dotenv -----
23 load_dotenv() # loads environment from .env into os.environ
24
25 # ----- logging -----
26 logging.basicConfig(
27     level = logging.INFO,
28     format = "%(asctime)s %(levelname)-8s %(message)s",
29     datefmt = "%H:%M:%S",
30 )
31
32 # ----- Flask setup -----
33 app = Flask(__name__, template_folder="templates", static_folder="static")
34
35 # ----- paths & DB file -----
36 BASE_DIR = os.path.abspath(os.path.dirname(__file__))
37 INSTANCE = os.path.join(BASE_DIR, "instance")
38 os.makedirs(INSTANCE, exist_ok=True)
39 DB_FILE = os.path.join(INSTANCE, "phishguard.db")
40
41 # ----- configuration -----
42 app.config.update(
43     SECRET_KEY = os.getenv("SECRET_KEY", "change-me"),
44     SQLALCHEMY_DATABASE_URI = f"sqlite:///DB_FILE",
45     SQLALCHEMY_TRACK_MODIFICATIONS = False,
46
47     MAIL_SERVER = os.getenv("MAIL_SERVER", "smtp.gmail.com"),
48     MAIL_PORT = int(os.getenv("MAIL_PORT", "587")),
49     MAIL_USE_TLS = os.getenv("MAIL_USE_TLS", "True") == "True",
50     MAIL_USERNAME = os.getenv("MAIL_USERNAME", ""),
51     MAIL_PASSWORD = os.getenv("MAIL_PASSWORD", ""),
52     MAIL_DEFAULT_SENDER = os.getenv("MAIL_DEFAULT_SENDER",
53                                     "PhishGuard Support <no-reply@example.com>"),
54 )
55
```

```

56 # ----- extensions -----
57 db = SQLAlchemy(app)
58 mail = Mail(app)
59 serializer = URLSafeTimedSerializer(app.config["SECRET_KEY"])
60 logging.info("→ Using database: %s", app.config["SQLALCHEMY_DATABASE_URI"])
61
62 # ----- models -----
63 class User(UserMixin, db.Model):
64     id = db.Column(db.Integer, primary_key=True)
65     email = db.Column(db.String(150), unique=True, nullable=False)
66     pw_hash = db.Column(db.String(256), nullable=False)
67     is_admin = db.Column(db.Boolean, default=False)
68
69 class Prediction(db.Model):
70     id = db.Column(db.Integer, primary_key=True)
71     user_id = db.Column(db.Integer, db.ForeignKey("user.id"))
72     input_type = db.Column(db.String(10)) # "email" | "url"
73     verdict = db.Column(db.String(12))
74     confidence = db.Column(db.Float)
75     timestamp = db.Column(db.DateTime,
76                           default=datetime.datetime.utcnow)
77
78 class ContactMessage(db.Model):
79     id = db.Column(db.Integer, primary_key=True)
80     name = db.Column(db.String(120))
81     email = db.Column(db.String(150))
82     body = db.Column(db.Text)
83     timestamp = db.Column(db.DateTime,

```

```

84                           default=datetime.datetime.utcnow)
85
86 # ----- auth setup -----
87 login_manager = LoginManager(app)
88 login_manager.login_view = "login"
89 login_manager.login_message = None
90
91 @login_manager.user_loader
92 def load_user(user_id: str) -> User | None:
93     return db.session.get(User, int(user_id))
94
95 def admin_required(fn: Callable):
96     @functools.wraps(fn)
97     def wrapper(*a, **kw):
98         if not current_user.is_authenticated or not current_user.is_admin:
99             return abort(403)
100         return fn(*a, **kw)
101     return wrapper
102
103 # ----- password-reset helpers -----
104 def send_reset_email(to_email: str):
105     token = serializer.dumps(to_email, salt="password-reset")
106     link = url_for("reset_token", token=token, _external=True)
107     msg = Message("PhishGuard Password Reset", recipients=[to_email])
108     msg.body = f"""Hello,
109

```

## Step 1 – Environment Setup & Imports

```
[1]: # Step 1.1: (Optional) Install any extra packages you need
# !pip install tensorflow scikit-learn

# Step 1.2: Imports
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report, accuracy_score

import tensorflow as tf
from tensorflow.keras import layers, models, Input, Model
from tensorflow.keras.callbacks import EarlyStopping
```

2025-06-14 07:19:45.688714: E external/local\_xla/xla/stream\_executor/cuda/cuda\_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered  
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR  
E0000 00:00:1749885585.889160 35 cuda\_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered  
E0000 00:00:1749885585.946819 35 cuda\_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered

## Step 2 – Load & Preprocess Data

### Step 4 – Build Base Model B: Deeper Network with Batch Normalization

```
[4]: def build_model_b(input_dim):
    inp = Input(shape=(input_dim,), name='input_b')

    # First dense block
    x = layers.Dense(128)(inp)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.Dropout(0.4)(x)

    # Second dense block
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.3)(x)

    # Third dense block
    x = layers.Dense(32)(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    # Output layer
    out = layers.Dense(1, activation='sigmoid', name='output_b')(x)

    model = Model(inputs=inp, outputs=out, name='model_b')
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
        loss='binary_crossentropy',
        metrics=['accuracy']
    )
    return model
```

```
import tensorflow as tf
from tensorflow.keras.layers import (
    Input, Embedding, Conv1D, GlobalMaxPooling1D,
    Dense, Dropout, Bidirectional, LSTM, concatenate
)
from tensorflow.keras.models import Model

# — Step 7a: Hyperparameter definitions —
VOCAB_SIZE = len(tokenizer.word_index) + 1 # Vocabulary size (+1 for padding token)
MAX_SEQ_LEN = X_train_os.shape[1] # Sequence length (e.g., 500)
EMBED_DIM = 100 # Embedding dimension

print(f"VOCAB_SIZE={VOCAB_SIZE}, MAX_SEQ_LEN={MAX_SEQ_LEN}, EMBED_DIM={EMBED_DIM}")

# — Step 7b: Model-building function definitions —

def build_cnn():
    inp = Input(shape=(MAX_SEQ_LEN,))
    x = Embedding(VOCAB_SIZE, EMBED_DIM, input_length=MAX_SEQ_LEN)(inp)
    x = Conv1D(128, 5, activation='relu')(x)
    x = GlobalMaxPooling1D()(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.5)(x)
    out = Dense(1, activation='sigmoid')(x)
    model = Model(inputs=inp, outputs=out)
```

```

        out = Dense(1, activation='sigmoid')(x)
        model = Model(inputs=inp, outputs=out)
        model.compile(
            optimizer='adam',
            loss='binary_crossentropy',
            metrics=['accuracy', tf.keras.metrics.AUC(name='auc')]
        )
        return model

def build_bilstm():
    inp = Input(shape=(MAX_SEQ_LEN,))
    x = Embedding(VOCAB_SIZE, EMBED_DIM, input_length=MAX_SEQ_LEN)(inp)
    x = Bidirectional(LSTM(64))(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.5)(x)
    out = Dense(1, activation='sigmoid')(x)
    model = Model(inputs=inp, outputs=out)
    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy', tf.keras.metrics.AUC(name='auc')]
    )
    return model

def build_hybrid():
    inp = Input(shape=(MAX_SEQ_LEN,))
    emb = Embedding(VOCAB_SIZE, EMBED_DIM, input_length=MAX_SEQ_LEN)(inp)
    emb = Embedding(VOCAB_SIZE, EMBED_DIM, input_length=MAX_SEQ_LEN)(inp)
    # CNN branch
    c = Conv1D(128, 5, activation='relu')(emb)
    c = GlobalMaxPooling1D()(c)
    # Bi-LSTM branch
    l = Bidirectional(LSTM(64))(emb)
    # Merge branches
    mrg = concatenate([c, l])
    d = Dense(64, activation='relu')(mrg)
    d = Dropout(0.5)(d)
    out = Dense(1, activation='sigmoid')(d)
    model = Model(inputs=inp, outputs=out)
    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy', tf.keras.metrics.AUC(name='auc')]
    )
    return model

print("Model-building functions defined: build_cnn(), build_bilstm(), build_hybrid()")

VOCAB_SIZE=34602, MAX_SEQ_LEN=500, EMBED_DIM=100
Model-building functions defined: build_cnn(), build_bilstm(), build_hybrid()

```

Here is the complete source code link

<https://drive.google.com/drive/folders/1MXDFtx3D-X4MgrD1kEYPQjwnnptqtzOH?usp=sharing>