# Saima Sultana

## Waleed_changes

The_2025
The
Shaheed Zulfiqar Ali Bhutto Institute of Science & Technology, Hyderabad

### Document Details

Submission ID
trn:oid:::1:3445475674

Submission Date
Dec 15, 2025, 7:09 PM GMT+5

Download Date
Dec 15, 2025, 7:59 PM GMT+5

File Name
waleed__Changes__1.pdf

File Size
2.6 MB

82 Pages

17,173 Words

98,204 Characters

# *% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

**Caution: Review required.**

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.



## Frequently Asked Questions

**How should I interpret Turnitin's AI writing percentage and false positives?**
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

**What does 'qualifying text' mean?**
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

# Social Sentry:
# AI-Based Cyberattack Detection of Phishing Links and Emails


Muhammad Waleed Raza
**01-136221-023**

Huzaifa Inam
**01-136221-009**

Supervisor: Dr. Sadia Nazim (Senior Lecturer)

**Bachelor of Science in Artificial Intelligence**

Department of Computer Science
Bahria University, Islamabad

December 2025

# Certificate

We accept the work contained in the report titled "SOCIAL SENTRY:AI-BASED CYBERATTACK DETECTION OF PHISHING LINKS AND EMAILS" written by MUHAMMMAD WALEED RAZA AND HUZAIFA INAM as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Artificial Intelligence.

**Approved by:**

**Supervisor:** Dr. Sadia Nazim (Senior Lecturer)

---

**Internal Examiner:** Dr. ABC (Assistant Professor)

---

**External Examiner:** Dr. XYZ (Professor)

---

**Project Coordinator:** Dr. XYZ (Professor)

---

**Head of the Department:** Dr. Muhammad Khurram Ehsan (Sr. Associate Professor)

---

December, 2025

# Abstract

Phishing is among the most widespread and harmful cyber threats, exploiting human trust to obtain credentials, financial data, and confidential information. This project introduces Social Sentry, an intelligent phishing detection system designed to identify malicious URLs and email content using modern deep learning techniques. The system incorporates two dedicated datasets: one consisting of phishing and legitimate URLs, and another containing phishing and benign email samples. These datasets enable Social Sentry to learn distinct patterns associated with both web-based and text-based phishing attacks.

Over time, phishing has evolved into sophisticated campaigns utilizing automation, AI-generated text, cloned websites, and highly deceptive communication patterns. As traditional blacklist-based or rule-driven systems struggle to detect such modern threats, advanced machine-learning models capable of adaptive and context-aware analysis have become essential.

The integrated platform includes URL analysis powered by a TabNet classifier and email phishing detection implemented through a Wide and Deep neural network, enabling the system to produce accurate predictions through feature-aware learning. A Python Django REST backend manages prediction requests, authentication, logging, and access to machine-learning models, while a responsive React frontend delivers intuitive real-time interaction.

To promote safer user decision-making, Social Sentry incorporates precautionary measures that alert users when a prediction is classified as phishing. Instead of integrating any external explanation API, the system provides clear warnings, recommended safety actions, and contextual cues to help users understand potential risks associated with suspicious URLs or emails. A continuous feedback loop allows users to validate results and contribute new samples for model retraining. Experimental evaluation demonstrates strong performance across both datasets, with high AUC scores and reliable real-time inference.

Overall, Social Sentry provides a scalable, interpretable, and user-friendly phishing detection solution suitable for educational, corporate, and individual cybersecurity environments. Its modular design supports future enhancements such as browser extensions, multilingual detection, attachment analysis, and integration with additional threat-intelligence sources.

# Acknowledgment

We are deeply grateful to our supervisor, Dr. Sadia Nazim, for her steady guidance, timely feedback, and consistent encouragement. We also thank the faculty and peers who challenged our ideas and helped refine this work through their comments.

We are also thankful for the supportive environment in our department, which allowed us to have open discussions and learn from others. The experience we gained during this project has improved our understanding and helped prepare us for future studies and work. Above all, we are very grateful to Almighty Allah for giving us the strength, knowledge, and patience needed to complete this project successfully.

Finally, a special thank you to our families and friends for their constant support and trust in us during this project, especially to our parents whose support paved the way for this thesis to be possible. Their belief in our abilities continuously motivated us to persevere despite difficulties.

MUHAMMAD WALEED RAZA & HUZAIFA INAM
Bahria University
E-8, Islamabad, Pakistan

December 2025

**”It always seems impossible until it is done. No journey is without struggle, but persistence transforms impossibility into success.”**

**Nelson Mandela**



turnitin Page 7 of 84 - AI Writing Submission Submission ID trn:oid:::1:3445475674

# Contents

*Contents*

# Chapter 1

# Introduction

## 1.1 Background

Internet communication has altered the mode of information exchange among individuals and institutions in this world. Despite the fact that this evolution affects productivity positively, it has also opened new fronts to cybercrimes. Phishing is one of the most frequent and destructive threats to the cyber environment. In contrast to the traditional attacks, which utilize software vulnerabilities, phishing uses the vulnerabilities of humans by revealing the trust or reputable organizations in the form of trusted people. Spoofed emails, fake hyperlinks and cloned websites are the methods used by attackers to convince users to disclose sensitive information (Passwords, financial, or confidential corporate records).

Phishing has developed over the years and turned into complex, high-target and automation and artificial intelligence-enhanced campaigns using weakly disguised messages. Hackers are now creating look-alike URLs, quickly establishing temporary malicious domains, and creating plausible emails that closely resemble the patterns of communicating with organizations. These nefarious abilities have been enhanced by the development of generative language models, which allow the automatic generation of linguistically natural and context-specific phishing messages. Consequently, conventional methods of detection, which apply only key-word matching, blacklists, or rule-of-thumb filters, are not always effective in detecting new forms of phishing activities.

## 1.2 Problem Statement

The existing phishing detection methods have a number of weaknesses making them less effective in combating the advanced attack methods. Blacklist based mechanisms only make use of previously known malicious domains and are not able to identify new domain

names or dynamically evolving ones. Email scanners that are based on rules tend to pay attention to superficial media features such as text rather than more profound semantic clues and are therefore not effective at detecting advanced phishing.

Moreover, most machine learning-based detection systems are black-box models, which give out predictions without the reasons, so that they are less trusted by the users and can be applied only under few situations in security-sensitive areas. The other key weakness of the majority of the publicly available tools is that most are narrow in scope; they are specifically built to analyze either URLs or email text not both. Also, a lot of open-source systems do not have real-time performance, extensive logging, and an internal explainable AI. All these limitations prove that a more sophisticated, versatile, and open phishing detection ecosystem is needed.

## 1.3 Objectives

The main aim of this project is to come up with a unified phishing detection system called, SocialSentry, that will be able to detect both URLs and emails in a single platform. The system uses the state of the art deep learning models, namely TabNet in URL classification and Wide and Deep neural network in email analysis to ensure high detection accuracy and interpretability.

SocialSentry includes an explainability module so that model outputs become clear and comprehensible by the final users of the system. The platform is designed with a responsive user experience by use of React based front end and Django REST back end that can support secure authentication, real time processing and effective management of data. Altogether, the project will provide the scalable and modular security system that will facilitate the ability to learn constantly, add the feedback of users, and be deployed on the local and cloud platform.

## 1.4 Project Scope

The scope of this project is the whole phishing detecting process and its input processing and presentation. Users are allowed to enter URLs or email texts, which the system preprocesses, extract features, and classifies deep learning models. Predictions, scores of confidence, and explanatory summaries are shown in a user-friendly interface.

The backend supports safe data processing, demand validation, inferencing, authentication and storing in a database. The system keeps the knowledge of the past user queries and model responses to guarantee transparency and facilitate future enhancements.

Furthermore, we have explainability module that will give natural-language explana-

tions of the individual predictions so that users can understand the reasons why certain URLs or emails are deemed as suspicious. In order to maintain control and monitoring, administrative instruments may be utilized for examining logs and handling user accounts. The elements work together to build a comprehensive phishing detection system where precision, openness, and user-friendliness are the main priorities.

## 1.5 Development Cycle

SocialSentry utilizes a methodical and step-by-step workflow to secure a high level of accuracy, robustness, and real-time performance. The first phase of the development cycle is evidently the acquisition of phishing URLs and email data from trusted open-source repositories. The next step is to preprocess the data in such a way that noise is eliminated, the structure is normalized, and the relevant lexical and behavioral features that are important for correct phishing detection are extracted. The complete workflow is illustrated in Figure 1.1.

Machine learning models are trained after preprocessing, TabNet for URL classification and a Wide & Deep network for email analysis. The selection of these models was based on their respective strengths; TabNet is excellent with structured tabular data while the Wide & Deep architecture captures both the memorization and generalization patterns in the email text. The optimized models after being trained and validated are then deployed into the system's inference pipeline.

A feedback observation mechanism enables the system to identify misclassifications and improve performance through optional retraining. This dynamic architecture ensures that SocialSentry adapts to evolving phishing techniques and remains effective over time. The system documentation provides a visual representation of this workflow, highlighting the cyclical nature of data ingestion, training, deployment, and improvement.

## 1.6 Motivation

The development of SocialSentry was based on the booming pace of the appearance and the level of sophistication of the phishing attacks. With the growing use of cloud services, remote work settings, and web-based systems by organizations, attackers are still taking advantage of the trust of human factor, instead of technical weaknesses. Phishing is always listed as one of the primary access points to ransomware, identity theft, financial fraud, and massive data breaches in global cybersecurity reports.

Phishing is not only a problem of individuals but also a problem of businesses and a single successful attempt leads to compromises of whole networks. Contemporary attack-

ers use domain spoofing, homograph attacks with Unicode characters, cloning websites, conversational phishing, and even voice-based deepfaking impersonation. The developing techniques require the implementation of smart, dynamic and transparent detection systems- inspiring the creation of SocialSentry.

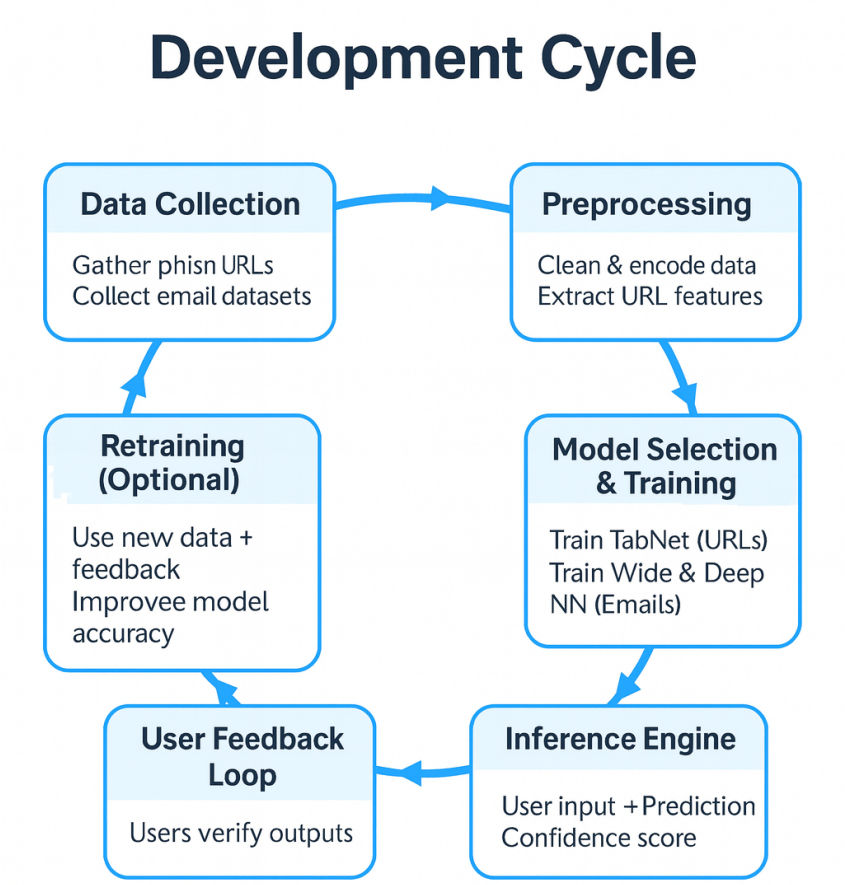


Figure 1.1: Development Cycle of the SocialSentry System

## 1.7 Need for Explainability in Security Applications

Modern cybersecurity tools need more than just detection accuracy. When the users know the reason behind a decision being made, they have more confidence in the systems, and this is particularly the case in dire circumstances like opening a suspicious email or URL. Most currently available tools cannot be explained, and people do not understand the reason behind content being flagged and get more likely to misinterpret.

SocialSentry is the solution to this since it relies on an inbuilt language model to create understandable explanations in human language. It identifies dangerous signs

that include abnormal subdomains, coded characters, novel names, mismatch of senders and use of urgency or rather urgency terms. In the case that a URL or email is classified as a phishing, the system will also offer cautionary steps, like not clicking on the link, not sharing credentials, or not checking the source, to help keep the user safe.

## 1.8 Justification for Model Selection

TabNet has been chosen to classify URLs because it is designed to work with structured tabular data, because it has a sequential attention mechanism, and because it can dynamically choose the most relevant features, including: token entropy, lexical structure and domain depth. This proves to be very useful in the analysis of the naturally structured URLs.

Wide & Deep model is selected as an email analysis because it is a hybrid architecture. The broad component recalls familiar phishing templates, and the deep component generalizes based on previously acquired representations, therefore, the familiar and new phishing attack can be tracked. The combination of these models offers great precision with little use of manual feature engineering.

## 1.9 Exclusions

To make the project viable and academically rigorous, various characteristics are beyond the limits of the project. The system is only useful in English and fails to scan file attachments like PDFs and documents. Personal inbox APIs (e.g., Gmail, Outlook) cannot be accessed directly because of privacy and compliance issues. Web crawling and threat intelligence collection of networks in large scales is also not provided because of the resource constraints. Also, the monitoring of the private messaging platforms (e.g., WhatsApp, Facebook, Instagram) is not considered in real-time, as these platforms do not allow automated access.

## 1.10 Summary

The chapter defined the reason why SocialSentry was created, as there is an increasing sophistication of phishing attacks and the insufficiency of current detection systems. It highlighted how the attackers use human behavior to an increasing degree creating false URLs and emails with the assistance of sophisticated AI-based programs-even more so than traditional filters and non-explainable machine learning models.

It was also in the chapter that the core objectives and scope of the project were presented and how SocialSentry unites the advanced models like TabNet and the Wide

& Deep network, as well as an explainability layer that is driven by a language model. The system is focused on real-time performance, safe backend functionality, user-friendly interaction, and transparent and ongoing logging.

These parts together provide a solid basis to the rest of the thesis, consisting of related literature, system requirements, system architecture, implementation strategy, and performance evaluation.

# Chapter 2

# Literature Review

## 2.1 Overview

Phishing is ever growing both in frequency and sophistication as it plays upon human vulnerabilities using deceiving, urgent, and emotional messages. Hackers create URLs, emails, and websites that look almost identical to real ones, making them extremely difficult to detect. Traditional rule-based systems cannot keep up with these adaptive threats, as attackers often change URLs, domains, and hosting patterns to avoid detection.

Recent research has shifted toward machine learning and deep learning methods, which can automatically learn harmful patterns from large amounts of data. These systems use feature-rich URL and email representations that can model complex relationships. They are more flexible and can generalize better compared to older static methods. In this chapter, the researchers review previous work in phishing detection and explain how these findings motivated the creation of Social Sentry, which uses TabNet for URLs, a Wide & Deep Neural Network for emails, a Django REST API for backend processing, and a React frontend for user interaction.

Moreover, the current AI-driven solutions provide the possibility of continuous refinement through the accumulation of new data with time. As a result, they remain powerful even if the perpetrators alter their techniques. Since the methods used for phishing are constantly changing, such flexible strategies have become indispensable in developing a trustworthy protection system.

## 2.2 Traditional Defense Mechanisms

The first measures employed to prevent phishing attacks were blacklists and whitelists along with heuristic filters. These methods gained popularity due to their simplicity of

implementation and their effectiveness in mitigating known dangers. Nevertheless, they have difficulties in uncovering novelties or attacks that are not detected, above all zero-day phishing cases. Furthermore, they become ineffective when the criminal uses URL masking or changes the email content as to not be caught by the rules. Phishing methods are getting smarter, and thus, the conventional methods have been rendered insufficient in the strong protection aspect.

In addition, these systems require constant manual updates, which makes them slow in responding to new threats. They cannot learn or improve automatically, giving attackers room to create new phishing tricks that can bypass fixed rules. Because of these limitations, researchers have moved toward using machine learning and deep learning models that offer smarter, more adaptive, and real-time phishing detection.

### 2.2.1 Blacklist and Whitelist Approaches

The blacklists store the malicious URLs and domains. Reporting Systems, including Google Safe Browsing and PhishTank, are premised on the ideas of crowd-sourcing, or centralized analysis. They work well with known threats but prove inefficient with new phishing URLs that have not been reported as of yet [5]. Whitelist is only applicable to restrict the access of access to trusted domains and not open internet environments.

### 2.2.2 Rule-Based and Heuristic Filters

User-based phishing detectors such as SpamAssassin are tools based on rules and function by examining established signals in a mail message. These indications usually include of the existence of a header anomaly, a suspicious or typically abused keywords, abnormal formatting pattern and structural deviations that are not in line with the acceptable level of communication. Although these systems were quite effective during the early period of detection of phishing, they have in the present-day era been strongly avoided. The attackers are able to distort the wordings, change formatting or modify metadata to a degree that does not cause a rule-based signatures [2]. The attackers can manipulate the wording, change formatting or modify metadata This continuing amalgamation of phishing letters makes such fixed guidelines less efficient, hence, scientists have begun experimenting with automated and data driven detection methods.

## 2.3 Machine Learning Approaches

The popularity of the machine learning techniques was founded on the principle of being able to learn the patterns in accordance with the aspects of the URL, HTML elements, email messages, webpage behaviors. In other works, such as that of Abu-Nimeh et al. [2]

algorithms such as SVM, Random Forest, and Logistic Regression have been compared, and the results of the algorithms had been found to be highly variant in terms of the quality and variety of features provided. However, Khonji et al. have mentioned that the application of manually designed features presents a constraint to the flexibility of the traditional ML models in cases where the attackers have devised novel ways of obfuscation [5]. Jain and Guha [6] also mentioned that despite the fact that ML-based phishing detection has a high rate of accuracy than rule-based ones, these models are not able to process high-dimensional and noisy data, which ultimately prompted the usage of deep learning.

## 2.4 Deep Learning Approaches

Deep learning models address a number of the limitations of the traditional machine-learning systems, which include learning hierarchical feature representations, computed automatically on raw or minimally processed data. Research by Basnet et al. 3 and Marchal et al. 4 has demonstrated that neural networks are useful in improving the rate of phishing when applied to email text, URL format, and webpage characteristics. Some of the benefits of deep learning in cybersecurity, as it was observed in surveys conducted by Sahoo et al. [1] and Chiew et al. [8] are the capability to automatically extract features and resist methods of obfuscating, improved generalization to novel phishing tools and the ability to model the complex non-linear relationships. These results had a great impact on the selection of the TabNet as structured URL data and a wide deep neural network as email-based phishing in the Social Sentry system.

## 2.5 Tabular Deep Learning with TabNet for URLs

Phishing datasets that are formulated using URLs are usually structured and tabular i.e. lexical features, domain age, the frequency of special characters, redirection behavior and WHOIS metadata. TabNet is a deep learning model that is directly trained to handle tabular data and it has several advantages over the traditional deep models that are used in this area. The sequential attention mechanism makes sure that it picks features sparsely, dynamically, and in each decision step process warrants on the most pertinent features. TabNet can also give interpretable decision pathways and can efficiently learn end-to-end and has a good performance on structured datasets. The features are in line with those of the literature of phishing detection, where the focus is on a multi-feature analysis and adaptive learning in the process of assessment of potentially malicious URLs

## 2.6 Wide & Deep Neural Networks for Emails

Phishing emails have to be detected and this requires the ability to identify the old phishing schemes and the new, unknown forms of messages. The Wide and Deep neural network structure is most suitable in such challenge because it incorporates two contradictory approaches to learning. The wide component stores the patterns that are frequently encountered e.g. common phishing text or structural hints and the deep component generalizes between the known in the form of inference of the latent semantic features of the contents of the email. The hybrid approach not only improves the detection accuracy and increases protection against new phishing varieties, which are the outcomes of the prior studies on the same topic of deep learning used to examine email-based threats

## 2.7 Explainability and Trust

The importance of explanation in the detection of phishing is rather high since the user and the analyst are expected to know the logic it used in delivering a decision to the user. To overcome this, Social Sentry gives recommendations on confidence by accompanying its binary predictions. There is also a clear indication whether a URL or an email is phishing or legit with the system giving a confidence level on such classification in each case. Such level of transparency would enable users and security analysts to estimate the certainty of the system, which leads to the trust in the output. Whether it is only a classification and confidence score is what makes the decision-making process easier and make sure the system is not concealed in regard to its forecasting ability.

## 2.8 Comparative Literature Analysis

The comparison of the large body of literature on major phishing detection works makes Table 2.1 display the very models utilized, the objective of the research, the inherent strengths, and the defined limitations. Such a comprehensive survey will help comprehend the most advanced condition by presenting various solutions, including machine learning and deep learning application to analyze URLs and email content. It aids in placing the proposed Social Sentry system into perspective, understanding its weaknesses and areas of enhancement, and finally justifying our choice of an integrated, multi-modal strategy of the detection.

The table compares key phishing detection studies, highlighting their methods and limitations. It also identifies research gaps that support the need for the Social Sentry system.

Table 2.1: Comparative Literature Review on Phishing Detection Techniques

| Reference | Models Used | Objective | Strengths | Limitations |
|---|---|---|---|---|
| Sahoo et al. [1] | ML classifiers (SVM, RF) | Survey of malware/phishing detection techniques | Provides broad overview of ML-based detection models | Lacks real-time performance evaluation |
| Abu-Nimeh et al. [2] | SVM, RF, Logistic Regression | Compare ML techniques for phishing detection | Highlights feature engineering importance | Limited semantic/-contextual analysis |
| Basnet et al. [3] | Neural Networks | Phishing email classification | Learns deep email patterns | Reduced interpretability (black-box) |
| Marchal et al. [4] | ML URL classifiers | Detect phishing sites | Good URL-target correlation analysis | Requires large URL datasets |
| Khonji et al. [5] | Survey | Review of phishing detection methods | Comprehensive taxonomy coverage | Descriptive survey without implementation |
| Jain & Guha [6] | ML models | Phishing attack detection | High URL/email accuracy | Dependence on handcrafted features |
| Rao & Pais [7] | ML classifiers | Detect phishing websites | Effective for structured datasets | No deep learning integration |
| Chiew et al. [8] | Survey | Attack types and technical approaches | Summarizes phishing vectors | Lacks practical deployment insights |

## 2.9 Research Gap

The existing phishing detection software can be said to have a few glaring weaknesses, which led to the development of Social Sentry. Most of the solutions either scan the URLs or categorize emails and, thus, they become broken and consequently cannot handle the multi-vector phishing attacks where a rogue link and a convincing email message work together. The other important gap is that the real time systems cannot be justified. The majority of the models are good at predictions, but they do not know how to describe an input as phishing and this is an issue that impedes the confidence of the users and its applicability on security settings.

## 2.10 How Social Sentry Responds

The constraints are addressed by the Social Sentry that analyses emails and URLs simultaneously with deep learning models. The system uses TabNet for URL detection, which automatically selects important features in structured data, and a Wide and Deep neural network for email phishing detection to combine memorization with strong generalization. Together, these two models ensure that all major phishing vectors are covered.

To overcome the lack of explainability in existing systems, Social Sentry generates clear, human-readable explanations for every prediction, helping users understand the reasoning behind each classification. The system is built using React on the front end and Django REST Framework on the back end, enabling real-time inference, secure authentication, and long-term storage of detection history. The overall analysis and identified research gaps are illustrated in Figure 2.1.

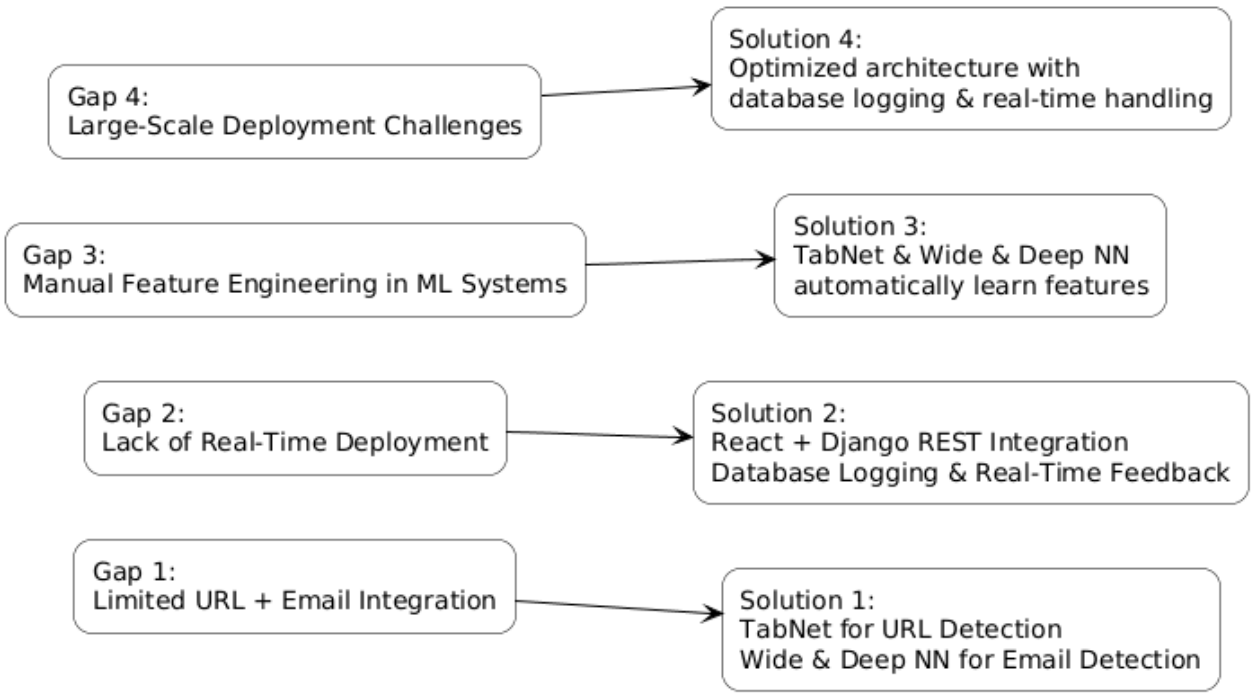


Figure 2.1: Research Gap Analysis of SocialSentry

The system promotes operational transparency by requiring every decision to have traceable reasoning steps, thereby increasing user trust even in the case of changing phishing attacks. Furthermore, Social Sentry shows safety measures like avoiding the link, not sharing credentials, and verifying the sender when a URL or email gets categorized as phishing in order to protect users without depend on the Gemini API or external explanation tools. The platform continues to be adaptable, allowing for continuous model updates and eventually better threat detection.

## 2.11 Summary

Detection of phishing is not only no longer a rule of thumb filter, but it can now apply more sophisticated deep learning algorithms to be trained on more complex URL and email patterns. On this development the Social Sentry makes use of TabNet to categorize URLs and a Wide and Deep neural network to examine emails, and built in explainability module to provide a clear explanation behind each prediction. A combination of such sophisticated components will create one and the same system of phishing detection, which will be extremely powerful and easy to read.

The chapter has also identified major gaps in the available researches and how the Social Sentry will address them. TabNet and Wide and Deep models render the feature engineering operations less significant and promote the ability of the system to discover new and emerging ways of phishing. Social Sentry draws a connection between the academic work and a practical solution by connecting explainable AI and executing the models within a fully deployable stack. The ideas and the conclusions of this chapter impacted the system design both directly and inspired the design process of the key points of the architecture making it robust and justified..

# Chapter 3

# Requirements Specification

## 3.1 Introduction

During the design or implementation of any technical system, it is mandatory to develop a full comprehension of the requirements of the systems before creation. The chapter outlines the functional and non-functional requirements of a phishing detection system called Social Sentry, which is developed using a React frontend, a Django REST backend, and a TabNet deep learning model. These requirements are used to come up with a reliable, scalable and user friendly security solution.

## 3.2 Existing System

Most traditional phishing detection systems are based on blacklists, rule-based methods, blocked URLs, or human-engineered features. Although these methods provide basic protection, they cannot keep up with new and changing phishing techniques. Attackers constantly hide harmful links, change domains, or use clever tricks that old systems fail to detect.

Another problem is that many existing solutions cannot clearly explain why a link or email is dangerous. Users only get a warning, but no helpful information, which makes it difficult for them to trust the system or learn from it.

Conventional methods have difficulties with real-time detection and also do not have the capacity for auto-adaptation. Hardly ever do they take advantage of user feedback for model enhancement, thus, getting obsolete very fast. Moreover, the human-machine interfaces of such systems are mostly basic, non-participatory, and opaque. The only notifications that are provided are the fundamental ones, in the form of binary alerts like “safe” or “unsafe,” but with no context. This situation results in misunderstanding, loss

of trust in the user, and a limitation in the capability of the system to assist in defense against cyber-attacks.

Moreover, such systems ignore user activities or habits, which might be a great help in spotting unusual behavior. They are also not capable of looking into the details of complicated emails, the attached files or the new type of phishing emails which are created by artificial intelligence. Users are still exposed to risks and companies still have to deal with more security threats due to these inadequacies.

## 3.3 Proposed System

Social Sentry, the proposed system, represents a highly advanced phishing detection model that incorporates deep learning (TabNet) together with the latest web technology and explainable AI methods. Unlike traditional methods, Social Sentry is able to handle both URLs and email content at the same time, thus providing real-time classification along with detailed feature-based explanations.

The system consists of a React-based frontend with a user-friendly interface combined with a Django REST backend that enables fast and smooth processes. With interpretability and history tracking incorporated in its models, users will be able to gain very helpful insights for every prediction which will consequently lead to increased transparency and trust. Moreover, the presence of an admin panel, reporting tools, and a feedback mechanism in Social Sentry is yet another side of the constant upgrading of the model support.

The proposed system is the solution that is the most powerful, the most intelligent and hence the most future prolific one to detect and overcome the weaknesses of the current systems by the use of a combination of scalable architecture, adaptive learning, and user-centered design.

## 3.4 Functional Requirements

The functional requirements define the minimum features and functions that Social Sentry should offer. The corresponding requirements allow users to detect phishing URLs in a fast, precise, and unambiguous way. These functionalities will make the system still practical, clear, and user-friendly for both common users and admins.

### 3.4.1 URL Input Acceptance

The system has to take a URL that a user provides for analysis. First, Input validation will clean and format the URL properly so that it can be processed. Second, the system must be able to identify and thus stop malformed or malicious input from slipping through

the validation checks. The input module should also assist users in such cases telling them if the URL they have given is incomplete or wrongly structured.

### 3.4.2 Phishing Classification

The TabNet deep learning model will have to categorize all URLs into Legitimate, Suspicious, or Phishing. The classification has to be done in real time, with a high degree of reliability and accuracy. The model will have to keep on getting better with the emergence of new threat patterns.

### 3.4.3 Explanation Generation

The system is expected to produce a distinct justification with the help of TabNet feature importance for every prediction. Additionally, this will bring in transparency and trust of the users to the system. The users will be made clear through the features which have had the greatest influence on the decision.

### 3.4.4 Real-Time Response

The system should produce forecasts in a matter of seconds at most in order to be suitable for interactive use without causing any disruption to the workflow. The response system must be improved in a way that it is able to cope with heavy loads in a smooth manner.

### 3.4.5 User Feedback Module

It is essential for the users to have the option of validating or altering the predictions made by the system. Besides, the feedback given should be kept for model retraining and dataset extension purposes in the future. Such a method enables the system to gradually transform and be in tune with the latest phishing tactics.

### 3.4.6 History Logging

The system is required to maintain a comprehensive record of previous scans, predictions, confidence levels, and user feedback. These logs will be available in the history panel. The saved records are beneficial for the administrators who want to go through the past threats and monitor the system's performance over a long period of time.

### 3.4.7 Admin Panel

It is necessary for administrators to have access to detection logs, analytics, user accounts, and system settings. They need to set up thresholds and behaviors. The admin panel

also has to enable the monitoring of model accuracy and system health.

### 3.4.8 Reporting Module

The system is required to produce summary reports that include detection statistics, performance measurements and long-term usage patterns. Besides, reports should be exportable for compliance or audit purposes.

### 3.4.9 Error Handling

The system needs to display unambiguous error messages corresponding to incorrect URLs, absent inputs, or server errors. The errors should lead the users directly to the problem's solution without any misunderstanding.

### 3.4.10 Session Management

The access to predictions, history logs, and admin features has to be given only to the authorized users by securing authentication with session tokens or JWT. To keep security, sessions should be terminated after a period of inactivity.

### 3.4.11 Accessibility

The user interface is required to be compatible with assistive technology, like screen readers, and to comply with accessibility regulations. This will make it possible for all users, even those with disabilities, to have an effective interaction with the system.

## 3.5 Non-Functional Requirements

### 3.5.1 Performance

Assigning a class to the URL should take no longer than the 2 seconds, even under a normal load condition; and the system must be able to use a consistent interface for an increasing load.

### 3.5.2 Security

The communication between the client and the server must be performed using HTTPS. To deter unauthorized access, sensitive data must be either encrypted or hashed. All modules must adhere to secure coding practices.

### 3.5.3 Scalability

A backend should be capable of handling a number of concurrent requests, and horizontally scalable in order to make room for future expansion.

### 3.5.4 Reliability

The performance level of the system should be no less than 99 percent uptime, and it should be able to deliver performance consistently under different workloads. The system must be able to handle high users without any interruption to the threat detecting services and it needs to be stable. In the event of a failure, automatic recovery mechanisms should be in place.

### 3.5.5 Maintainability

The modular architecture (React, Django REST, TabNet) should be able to sustain future changes and maintenance in the long run easily. Documentation should be of good quality to make it easier for the new developers to get trained.

### 3.5.6 Portability

The system should be able to operate across Windows, Linux, and macOS platforms. Deployment scripts should be universal so as to support distinct hosting environments.

### 3.5.7 Responsiveness

The frontend has to be completely responsive both on desktop and mobile. The UI elements need to change their size and position in a very user-friendly manner as per the different screen sizes.

## 3.6 Use Case Diagrams

There are typically 3 use cases for Social Sentry which are discussed respectively.

### 3.6.1 User-Side Use Case

This part brings up significant operational prerequisites for the system's good performance, such as input data validation, fast user feedback, and lucid and intelligible security analyst reports making. These features not only make operation more effective but also add strength to the security power of Social Sentry. The thorough description

of the interaction flow does more than just offering the application structure, it is also helping to keep the system development process open to both expected and unexpected results. Moreover, it enables the establishment of the core protection and response processes that are essential to Social Sentry's objective. The entire use case detailing all the steps from harmful link detection to management is represented in Table 3.1, which means a systematic approach that assures the coverage of all operations and assists the development teams in setting up strong error handling and maintaining system integrity under different situations.

The standard user interaction is centered around the process where a registered user inputs the URLs to phishing detection, obtains an explanation, and views the results recorded.

Table 3.1: Use Case UC1: URL Phishing Detection

| | |
|---|---|
| **Use Case ID** | UC1 |
| **Actor** | Registered User |
| **Description** | The user submits a URL to determine whether it is legitimate, suspicious, or phishing. The system analyzes the URL and returns the prediction along with a structured explanation. |
| **Precondition** | User must be authenticated. |
| **Postcondition** | The resulting prediction and explanation are stored in the user's history. |
| **Normal Flow** | • User logs in.<br>• User provides URL for analysis.<br>• System validates the input.<br>• Backend sends URL to model.<br>• System displays classification, explanation, and confidence.<br>• History is updated. |
| **Exception Flow** | • Invalid URL format.<br>• Model prediction error.<br>• Explanation service unavailable. |

The user use case is illustrated in a visual manner in Figure 3.1, where the whole process is summarized. The diagram gives a very clear understanding of the sequence of actions by portraying users' main activities along with the system's replies. Furthermore,

it presents the communication of the frontend and the backend during a normal detection phase, thus providing a lucid picture of the real-time phishing analysis process. The diagram is very important for the comprehension of the system, in which not only the asynchronous tasks but also the synchronized information exchanges necessary for stable service are included.

In addition, the flow diagram depicts the major points where data validation takes place and security flags get generated. The whole process enables the user to get real-time accurate and timely information regarding the security status of the examined link. The system's ability to keep the user informed and respond quickly has a positive impact on its efficiency as a threat's mitigation tool. The process further assists the users in gaining trust in the system's judgments by unambiguously indicating the role of each security check in the final result.

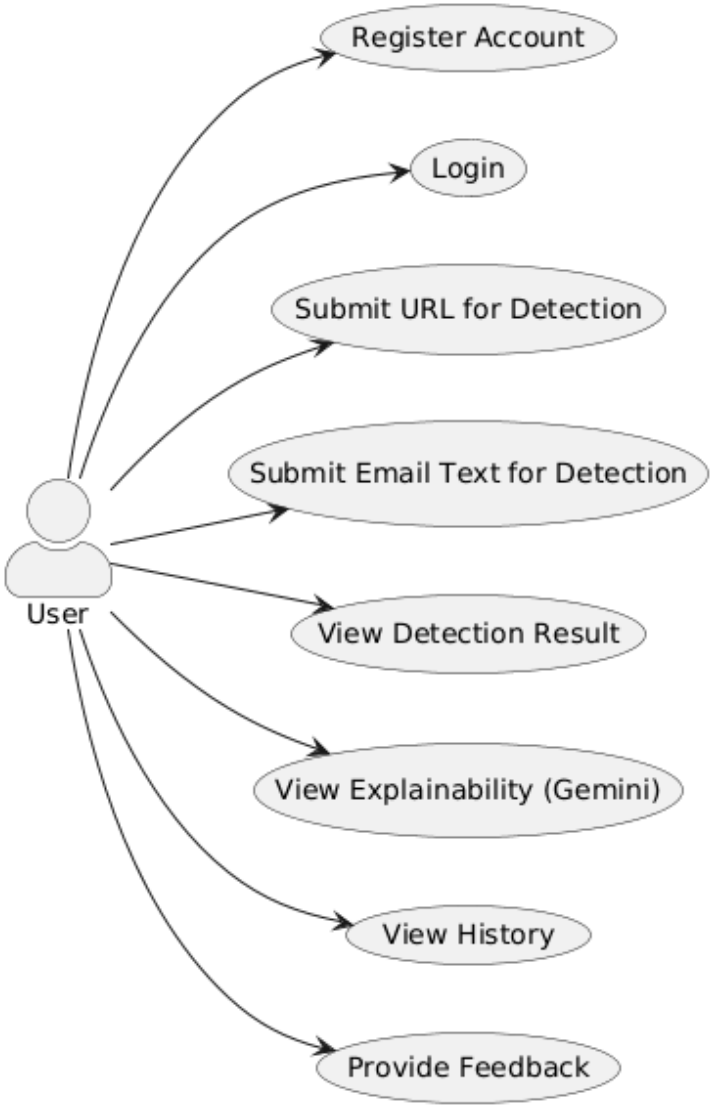


Figure 3.1: User-Side Use Case Diagram for Social Sentry

### 3.6.2 Admin-Side Use Case

The use case UC2 presented in Table 3.2 is particularly very important, as it deals with administrators' reviewing of phishing attempts, and such cases involve theft of credentials or targeted attacks. A systematic workflow is implemented in which the entire input sources, be it URLs or email texts, are processed in a uniform way, and the deep learning models predict the outcomes having provided the explanation outputs. The final classification and its explanation are securely logged, thus providing detailed record-keeping necessary for system audits and regulatory compliance. Centralized logging not only forms a strong forensic trail but also helps in rapid incident response and model detection optimization based on current threat patterns. In addition to this, proper documentation enables administrators to spot attack vectors that are being reused and to modify security policies in a corresponding manner. Moreover, the stored knowledge can also be used in new employee training as they will be given real examples of threat scenarios.

Table 3.2: Use Case UC2: Email Phishing Detection

| | |
|---|---|
| **Use Case ID** | UC2 |
| **Actor** | Registered User |
| **Description** | The user submits email text to be analyzed for phishing indicators. The system processes the text, generates a classification, and provides an interpretability explanation. |
| **Precondition** | User must be authenticated. |
| **Postcondition** | Prediction and explanation are stored in the user's history. |
| **Normal Flow** | • User logs in.<br>• User enters email text.<br>• System validates the input.<br>• Backend processes the email text via ML model.<br>• Prediction and explanation are displayed.<br>• History updated. |
| **Exception Flow** | • Empty input provided.<br>• Model or processing error.<br>• Explanation service unavailable. |

The administrator interface of the system is depicted in detail in Figure 3.2 below (adminusecase). The diagram highlights the administrator's rights, which include setting up the system, monitoring the user activity, and dealing with the detection logs. It also shows how the administration acts to form the back-office processes that allow easy and uninterrupted control and operation of the system. This diagram is a visualization that allows one to see the division of responsibilities and the access control being enforced over the whole system.

The figure also gives information about the interfaces for checking system health metrics and updating detection models, which are considered very crucial for maintaining high system reliability and performance. From this all-encompassing viewpoint, it becomes very clear that all the necessary controls are in place to manage and secure the system in a very effective way. Moreover, the figure indicates the processes like compliance report generation and routine data backups that have been put in place to make accountability and disaster recovery planning easier through the application of clear procedures.

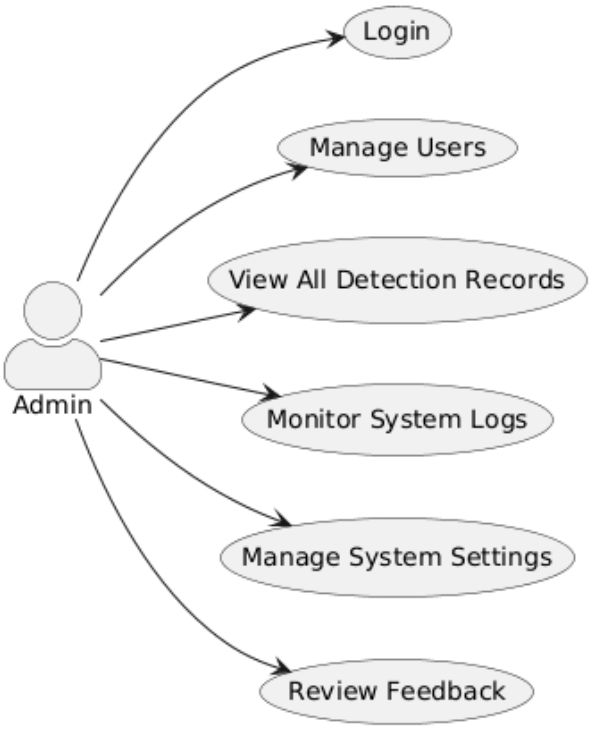


Figure 3.2: Admin-Side Use Case Diagram for Social Sentry

### 3.6.3 System-Level Use Case

Table 3.3 illustrates Use Case UC3 that demonstrates a process where users can retrieve and, at the same time, review the history of the stored detections. It explains the conditions, system actions, and hopefully, no exceptions. This use case is very significant since

it enables the user to recall the past detection of phishing and know the system's response during that time for each scan. It also helps the admins to observe long-term trends, confirm the total system performance, and recognize the phishing behaviors that are coming back. Telling users the past results in no time reaps the benefits of transparency, faster incident investigation, and insightful detection model refinement. Moreover, past records being in an organized manner helps in compliance audits, as the audit can require particular evidence of system activity.

The system, by offering absolute visibility of the previous detections, increases the trust and the security improvement on the platform. The organized history is also a means of better cooperation among the administrators because they all share the same view of detection events and can, for example, identify the false positives and negatives over time, leading to the more accurate calibration of the model. This use case, thus, guarantees that the detection system is getting smarter by learning from the users' real-world interactions and the threat data. The historical data, in a way, also reveals the emergence of attack patterns and gives the security team an edge in mounting a proactive defense by adjusting their measures. Figure 3.3 demonstrates the collaboration or rather the interaction of the frontend, backend, ML model, and database to perform all the core functions of the system.

Table 3.3: Use Case UC3: View Detection History

| | |
|---|---|
| **Use Case ID** | UC3 |
| **Actor** | Registered User |
| **Description** | The user views past scan results including predictions, confidence scores, timestamps, and explanations. |
| **Precondition** | User must be authenticated. |
| **Postcondition** | History records are displayed with filtering and review options. |
| **Normal Flow** | • User logs in.<br>• User navigates to the history section.<br>• System retrieves stored detection records.<br>• Records are displayed to the user. |
| **Exception Flow** | • No history records available.<br>• Database retrieval error. |

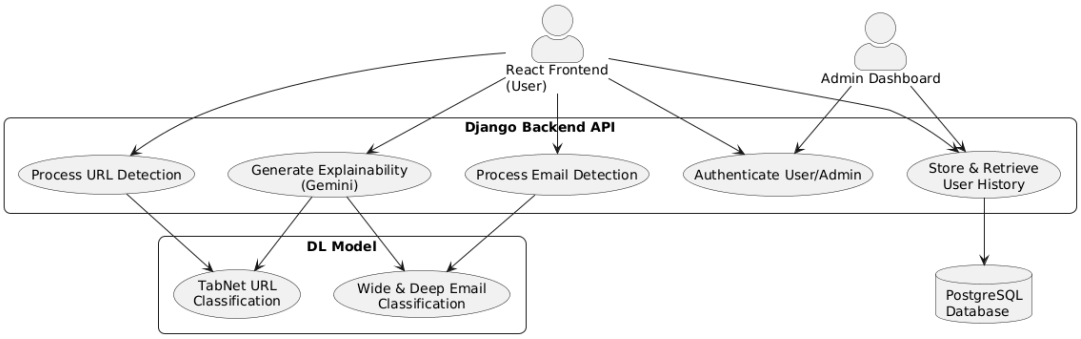


Figure 3.3: System-Level Use Case Diagram for Social Sentry

## 3.7 Software Requirements

The following table lists and explains all software components essential for implementing Social Sentry. It provides clarity regarding the purpose of each tool and framework used within the system. Table 3.4 (Table 3.4) summarizes the required software stack. These components ensure that the system remains scalable, secure, and efficient across all modules. Together, they form the technical foundation that enables reliable phishing detection and seamless user interaction.

Table 3.4: Primary Software Components for Social Sentry

| **Software Component** | **Role** |
|---|---|
| Python 3.10+ | Backend and TabNet inference |
| React.js | Frontend interface |
| Django + DRF | API services |
| DBSqLite3 | Storage |
| PyTorch-TabNet | Deep learning model |
| NumPy / Pandas | Data preprocessing |
| scikit-learn | Feature scaling |
| Git | Version control |

## 3.8 Hardware Requirements

The hardware requirements outline the low-level requirements to operate Social Sentry effectively when developing, testing, and deploying the system. The specifications will provide that the server, machine-learning models, and front-end interface can run smoothly without delays or resource constraints. Table 3.5 summarizes the hardware components and specifications needed. These guidelines are important to ensure the required processing power to conduct real-time deep learning (DL) inference and a sufficient amount of memory to process large datasets. This upstream planning will help avoid any

performance bottlenecks and provide scalability of the system as the number of users increases. Moreover, addressing these needs is crucial to the achievement of the support of simultaneous user requests and high availability requested by a highly important security tool.

Table 3.5: Hardware Requirements for Social Sentry

| **Component** | **Specification** |
|---|---|
| Processor | Intel CPU (Recommended: quad-core or above for DL inference stability) |
| RAM | Minimum 8 GB (Higher memory improves model loading and API responsiveness) |
| Storage | At least 1 GB free space for datasets, logs, and system files |
| Network | Stable internet connection for API communication and data retrieval |

## 3.9 Summary

This chapter laid the ground work of the features of the creation of the Social Sentry. It has described the constraints of the current phishing detection mechanisms, presented the solution offered, and established the functional and non-functional requirements to have a platform that could be built to be reliable and scalable. More elaborate use case diagrams also showed the interaction of users and administrators with the system, as the software and hardware specifications defined the bare components that are needed to be implemented. These requirements combined create an easy-to-follow blueprint that supports the design, development and implementation of. the Social Sentry phishing detector system..

# Chapter 4

# System Design

## 4.1 Introduction

This chapter presents the system design of the proposed phishing-detection platform, Social Sentry, which analyzes URLs and email messages using machine-learning models. The system is built on a Django REST backend with dedicated URL and email detectors running on separate components, supported by an explainability layer that generates clear, human-readable interpretations.

This system is designed as a modular system wherein each module is permitted to operate independently, but to interact with the rest of the modules through well defined interfaces. This makes the system upkeepable and easily extendable in the subsequent versions.

## 4.2 Overall Architecture

Social Sentry uses a simplified three-layer architecture:

- **Presentation Layer:** A lightweight frontend interface where users submit URLs or email text and view predictions, explanations, and history.
- **Backend Layer (Django REST Framework):** Implements authentication, phishing detection APIs, history management, and communication with machine-learning services.
- **DL & Explainability Layer:** Contains the phishing classifiers and an explanation engine that generates human-readable interpretations along with precautionary safety measures for detected phishing content.

As shown in Figure 4.1, the architecture diagram visually represents how the frontend, backend, and machine-learning services interact. It shows the flow of data from the user to the phishing detectors, explanation module, and finally to the database. This visual overview clarifies how each layer collaborates to deliver real-time phishing detection.

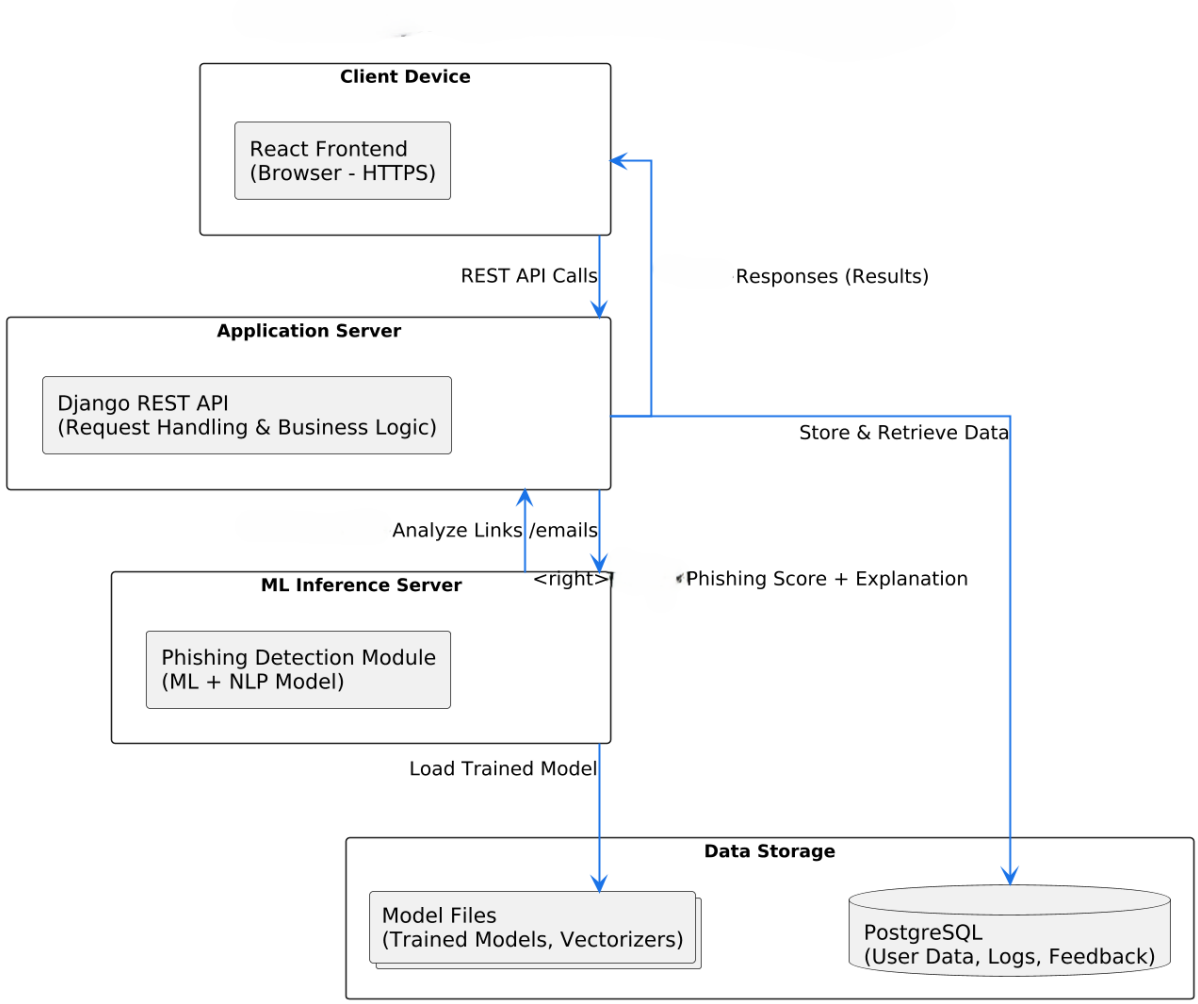


Figure 4.1: Implemented System Architecture of Social Sentry

## 4.3 Implemented API Endpoints

The following API endpoints are actually implemented and used in the system.

### 4.3.1 Authentication and Accounts

- **POST /api/accounts/register/** – Register a new user.
- **POST /api/accounts/login/** – Obtain JWT access and refresh tokens.
- **POST /api/accounts/token/refresh/** – Refresh the JWT access token.

### 4.3.2 Phishing Detection

- **POST /test/check_url/** – Detect phishing or legitimate URLs.
- **POST /test/check_email/** – Detect phishing or legitimate email content.

### 4.3.3 User History

- **POST /api/history/save-history/** – Save a user's detection record.
- **GET /api/history/get-history/** – Fetch all detection history for a user.

These endpoints form the communication backbone between the frontend, backend, and DL components.

## 4.4 Component Diagram

The implemented system contains only the following real components. Unused modules such as admin dashboards, training pipelines, workers, event buses, and cloud orchestration elements have been removed.

- **User Interface (UI):** Sends input (URL or email text) and displays prediction results, explanations, and history.
- **Authentication Module:** Handles registration, login, JWT generation, and token refreshing.
- **Detection Module:** Provides API endpoints for URL and email phishing detection.
- **DL Models:** Tabnet for URL and Wide  Deep Neural Network for Email.
- **Explanation Module:** Generates short, human-readable explanations and provides precautionary safety measures when phishing content is detected.
- **History Module:** Saves and retrieves user detection history.
- **Database Module:** Stores users, predictions, explanations, timestamps, and history records.

According to Figure 4.2 below, the component diagram shows the key modules of the system and how they interact with each other driven by well-defined interfaces. It isolates user-facing layers together with backend logic and machine-learning services and offers a

structured picture of how the inner organization of the system can be organized. This design is in a modular form, which leads to the independent development and testing of each module, which is important to maintainability and reliability of the system. This strict separation also means that the deep learning models do not need to be changed when the frontend interface is updated to evolve the system in the long term and upscale.

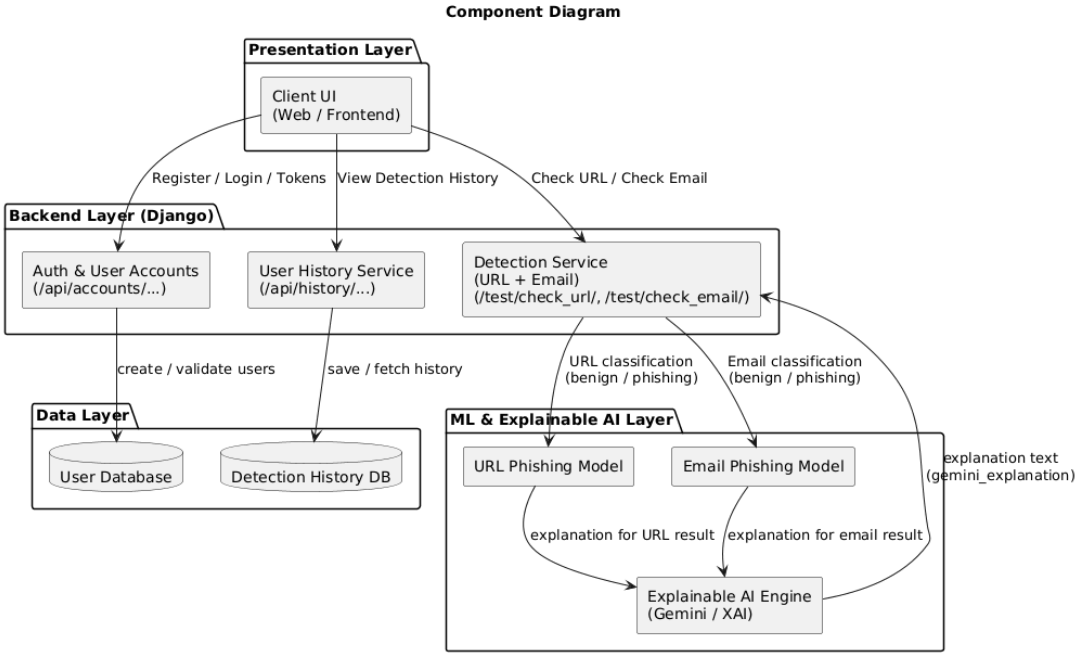


Figure 4.2: Implemented Component Diagram of Social Sentry

## 4.5 Inter-Component Communication

Only communication methods that are implemented in the system are included:

- **RESTful API Calls:** Frontend communicates with Django backend using JSON requests over HTTPS.
- **Internal Python Function Calls:** Backend services directly call preprocessing functions, model inference functions, and the explanation generator.
- **Database Transactions:** Simple create/read operations for user data and history retrieval.

No message queues, event-driven systems, or background workers are part of the implemented design and have been removed.

## 4.6 Database Design

The simplified database includes:

1. **Users:** Authentication and account details.
2. **Detections:** URL or email inputs and their results.
3. **Explanations:** Provides precautionary safety measures.
4. **History:** Past detection records linked to users.

As shown in Figure 4.3, the ER diagram visually shows how the main entities—Users, Detections, Explanations, and History are related. It reflects only the operational tables used in the deployed system, ensuring the diagram accurately matches the database implementation.

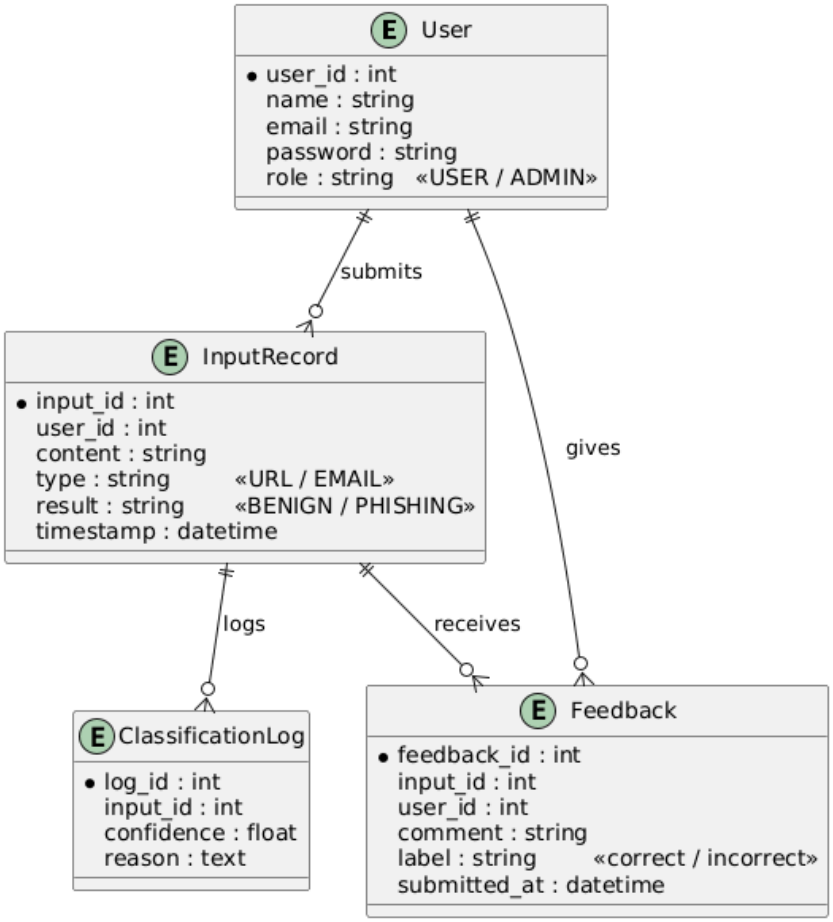


Figure 4.3: ER Diagram for the Implemented Database Design

Only tables used in the actual implementation have been retained. Entities such as sessions, model metadata, and retraining data (not implemented) have been removed.

## 4.7 Sequence Diagram of Implemented Flow

The following sequence represents the actual implemented flow of the system.

1. User logs in and receives a JWT token.
2. User submits a URL or email text for analysis.
3. Backend validates the request and forwards it to the DL model.
4. DL model extracts features and produces a prediction.
5. Backend sends the input to the explanation module.
6. Generates short, human-readable explanations of precautionary measures.
7. The result, confidence, explanation, and timestamp are saved.
8. Backend returns result + explanation to the user interface.

As illustrated in Figure 4.4, the sequence diagram provides a step-by-step visualization of the system workflow. It shows interactions between the UI, backend, DL model, and explanation generator, clearly outlining how real-time phishing predictions are processed and returned.

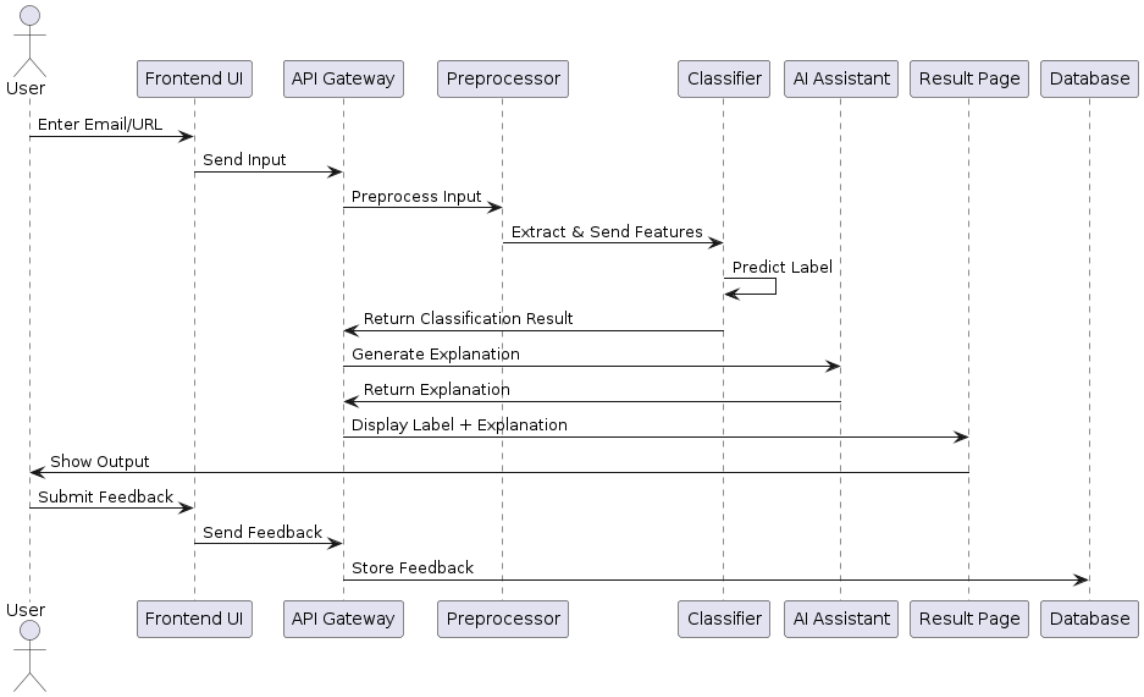


Figure 4.4: Sequence Diagram of Implemented Phishing Detection Workflow

## 4.8 Methodology

The phishing-detection software has been designed in a systematic and progressive engineering process that has been created to offer strength, extensibility, and usability. The methodology has eight major steps and these steps are requirement analysis, datasets selection, data preprocessing, model development, model training and evaluation, system design, system integration and testing, and deployment. These stages ensure that each component of the system is developed, refined, and validated before moving to the next phase. This structured approach also helps maintain consistency and reduces errors throughout the development lifecycle. The phases are as described below..

### 4.8.1 Requirement Analysis

Stage 1 was dedicated to the system objectives, phishing problems, and working restrictions. Among the primary requirements, there existed adequate real-time phishing prediction, which had the capacity to process. input types (URL and email), good processing on the back-end and a combined methodology with a clean front-end interface.

### 4.8.2 Dataset Selection

The datasets of phishing were researched publicly to insure the choice of credible and diverse. sources. The final data set was the phishing URLs and email based phishing records, which assured the large range of attack vectors. The accuracy of the information, money, and representativeness were put to check before they were included.

### 4.8.3 Data Preprocessing

Data preprocessing involved cleaning of raw data, coding of discrete and textual data, deriving discriminative features, and class imbalance. The dataset was made ready using the. be such that it uses methods such as tokenization, vectorization, normalization and resampling. learned in machine learning pipelines.

### 4.8.4 Model Development

A number of machine learning and deep learning models were used and compared. Algorithms were chosen based on their capacity to operate on textual properties and non-linear relationships that are typically seen in phishing data. The most successful model was put into practice. Additional tuning and experimentation were performed to maximize accuracy and reduce false classifications. This ensured that the final selected model offered both strong performance and reliability for real-world deployment.

### 4.8.5 Model Training and Evaluation

The selected model was to deal with the unbalanced phishing and legitimate samples. class weighted training. The ability to generalize was determined by computing. accuracy and F1-score are performance measures. Hyper parameters were fine-tuned. to obtain more realistic predictive outcomes on a real world situation.

### 4.8.6 System Design

The system architecture was intended to bring together data transport, processing modules and storage layers. This was including the definition of the backend API, the parsing of the frontend. interaction, database schema and prediction request pipeline. The focus was made on flexibility and simple servicing.

### 4.8.7 System Integration

It was implemented through the assistance of the incorporation of a React-based frontend and a Django. backend. All the components are exchanged through the RESTful endpoints to create scalable and secure. functionality.

### 4.8.8 Testing and Deployment

It was implemented using the assistance of the combination of a React-based frontend and Django backend. The architecture had strong provisions to accommodate more sophisticated prediction services and these are the threat intelligence lookups and sandboxing of suspicious inputs. All the elements are exchanged through RESTful endpoints in order to provide scalable and secure functionality. Moreover, the measures such as strict input validation and automated analysis were implemented whenever a user provides a suspicious URL or email, which guaranteed the reliability and integrity of the system.

**Figure 4.5** is a workflow diagram that includes the entire nine stages of methodology that will be utilized in developing and deploying the phishing detection system. It starts with Requirement Analysis, during which system requirements, particular phishing issues, and the general project objectives are determined. It is then succeeded by Dataset Selection which involves exploration of public datasets to select appropriate phishing URL and email datasets required to be used to train. The initial major stages end with Data Preprocessing that involves data cleaning, encoding and feature extraction, and the issues of class imbalance are dealt with to prepare data to be used in the learning stages.

The next phases are concerned with the essences of machine learning. Model Development entails trial and error of different Deep Learning (DL) models to narrow down on the most appropriate model to deploy. The chosen model is then subjected to the Model Training & Evaluation where it is trained on a variety of techniques such as class weighting and its performance is thoroughly assessed with the help of such standard metrics as accuracy and F1-score. After the effective design and testing of the core detection mechanism, the System Design phase is initiated where the architecture of the system, components structure, and process flow, and the general database schema are defined.

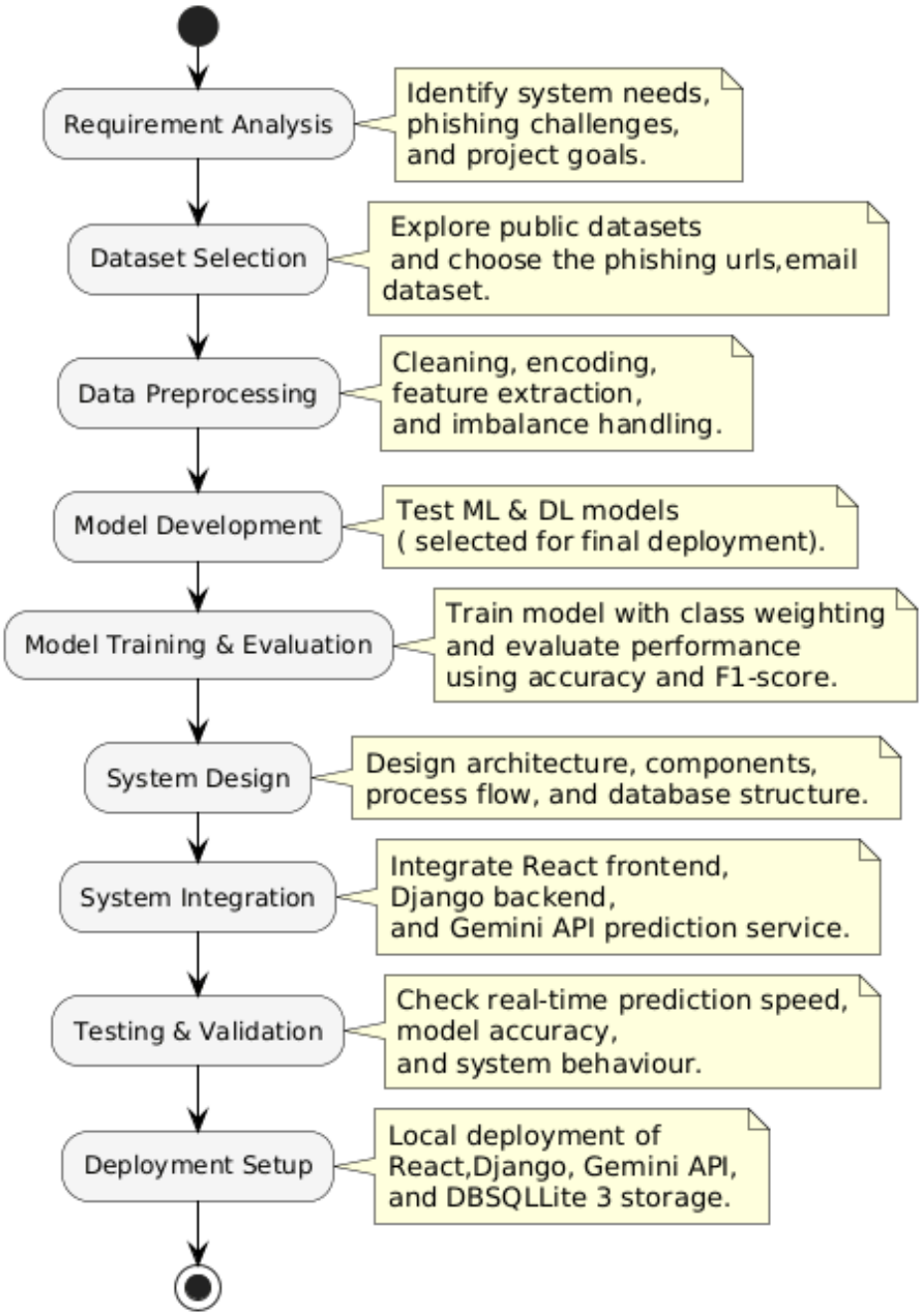


Figure 4.5: Workflow diagram illustrating the methodology pipeline.

The last three phases outline the realisation of the system and how it is ready to be used. System Integration entails integrating the main technology stack elements, which are the React frontend and Django backend and the machine-learning prediction services to analyze the URLs and emails. The system is then comprehensively Tested and Validated to test the speed of real time predictions, the model accuracy, and the general behaviour of the system in different situations. The last step in the pipeline is Deployment Setup which defines local deployment of React application, Django backend, DL prediction services and DBSQLite3 storage components.

## 4.9 High-Level System Overview

As shown in Figure 4.6, this high-level view summarizes the entire flow of data—from user input to backend processing, AI-based explanation, and storage in the database. It helps clarify how all modules work together to deliver the final prediction to the user.

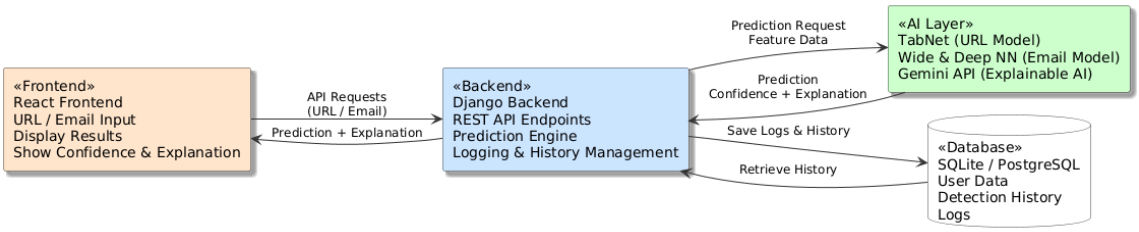


Figure 4.6: High-Level System Overview of SocialSentry

## 4.10 GUI Design

Social Sentry had a Graphical User Interface (GUI) that was designed with a focus on. transparency, simplicity, and a contemporary aesthetic. Each interface is of a recurring nature. dark-purple visual and smooth gradients, rounded, and legible typography. The interface is complete responsive, which guarantees its use with desktops, laptops, and mobile devices.

Each of the implemented screens is presented below with a description. of its intent and graphical design.

### 4.10.1 Landing Page

The landing page serves as the entry point for both users and administrators. It provides two login options and introduces the system with a visually appealing hero section, as illustrated in **Figure 4.7**.

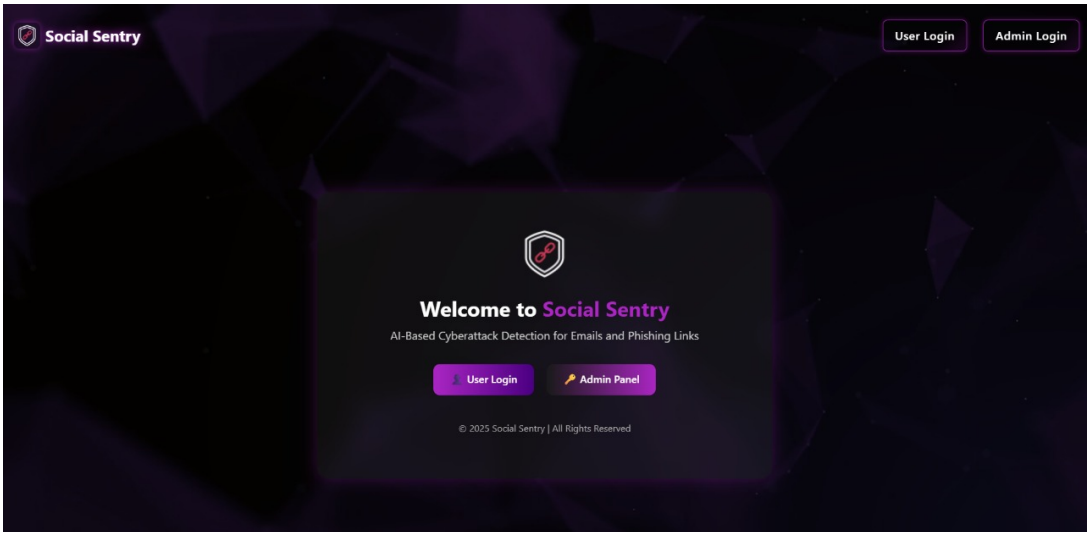


Figure 4.7: Landing Page Interface of Social Sentry

### 4.10.2 User Login Screen

The user login page allows authenticated access to the detection system. It includes fields for email and password, along with a link to create a new account, as shown in **Figure 4.8**.

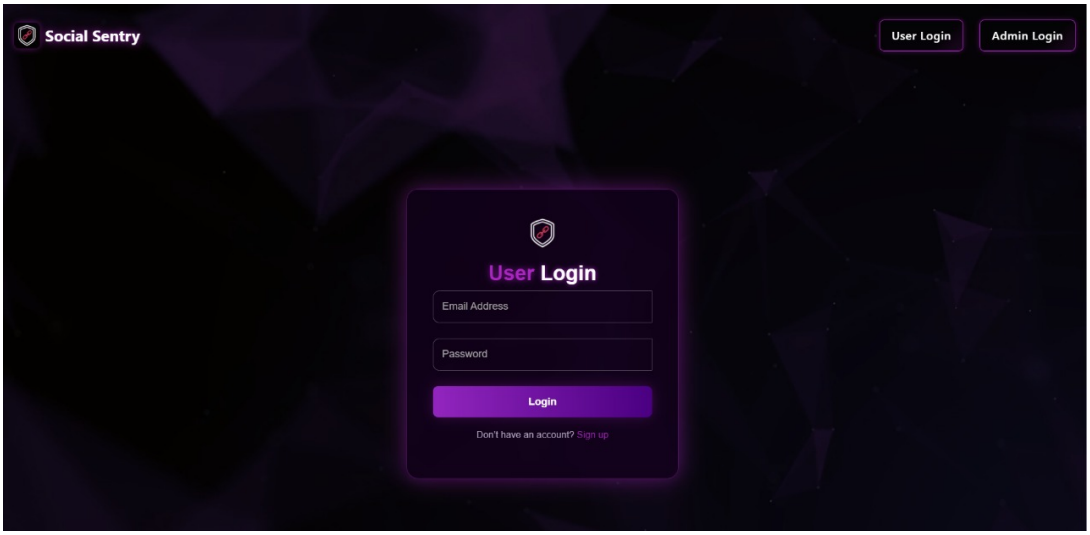


Figure 4.8: User Login Screen

### 4.10.3 User Registration Screen

The registration interface enables new users to create an account by entering their full name, email address, and password. The design maintains a clean and minimalist form structure, as depicted in **Figure 4.9**.

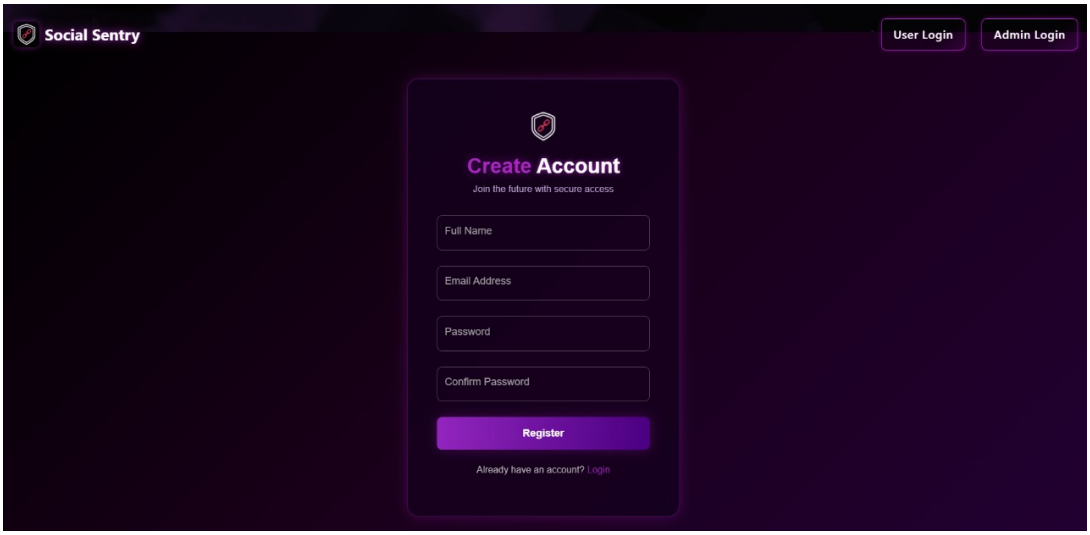


Figure 4.9: User Registration Screen

### 4.10.4 AI-Powered Detector Interface

This is the primary screen used for phishing analysis. Users input a **URL or email content**, and the system displays the **prediction, detection confidence, and explanation**, as illustrated in **Figure 4.10**.

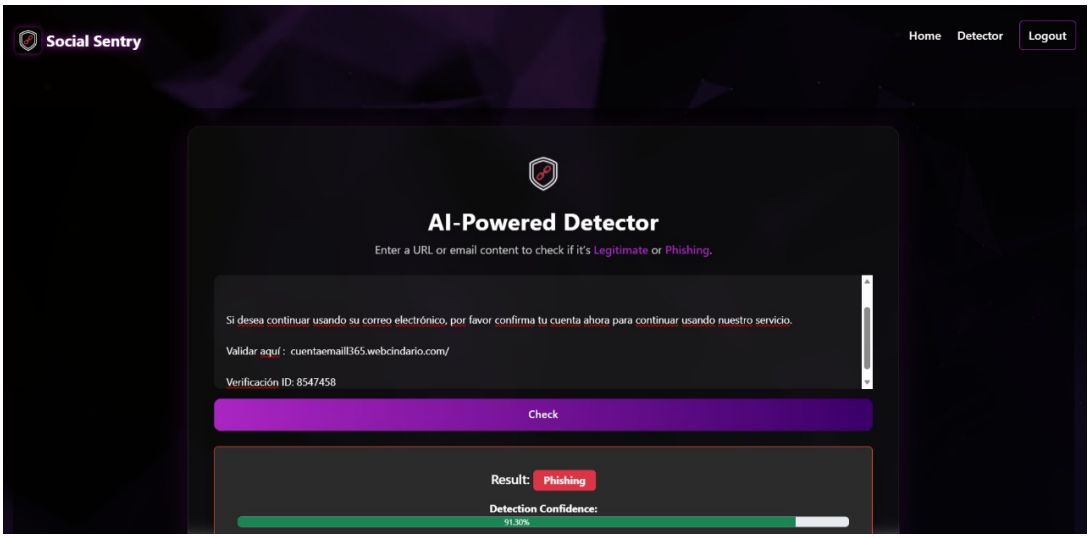


Figure 4.10: AI-Powered Phishing Detector Interface

### 4.10.5 Admin Panel – Scan History

Administrators can view historical detection records, phishing statistics, and filter the entries by timestamps such as last 10 records, last week, or a custom range. The interface also supports exporting data to Excel, as shown in **Figure 4.11**.

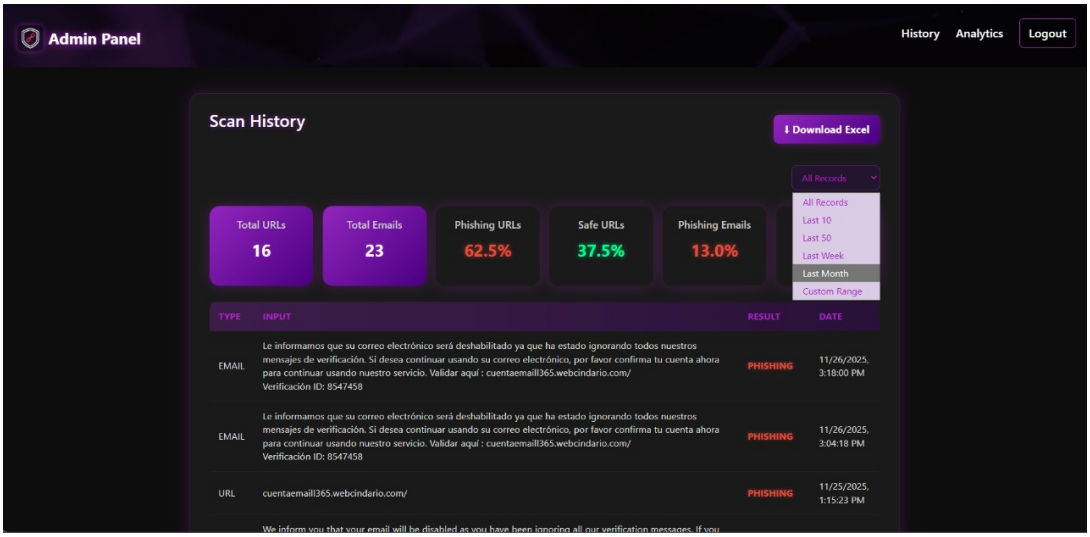


Figure 4.11: Admin Panel – Scan History Interface

### 4.10.6 Admin Analytics Dashboard

The analytics dashboard provides high-level insights into system performance, including counts of phishing vs legitimate URLs and emails, as well as accuracy trends of different DL models over training epochs, as illustrated in **Figure 4.12**.

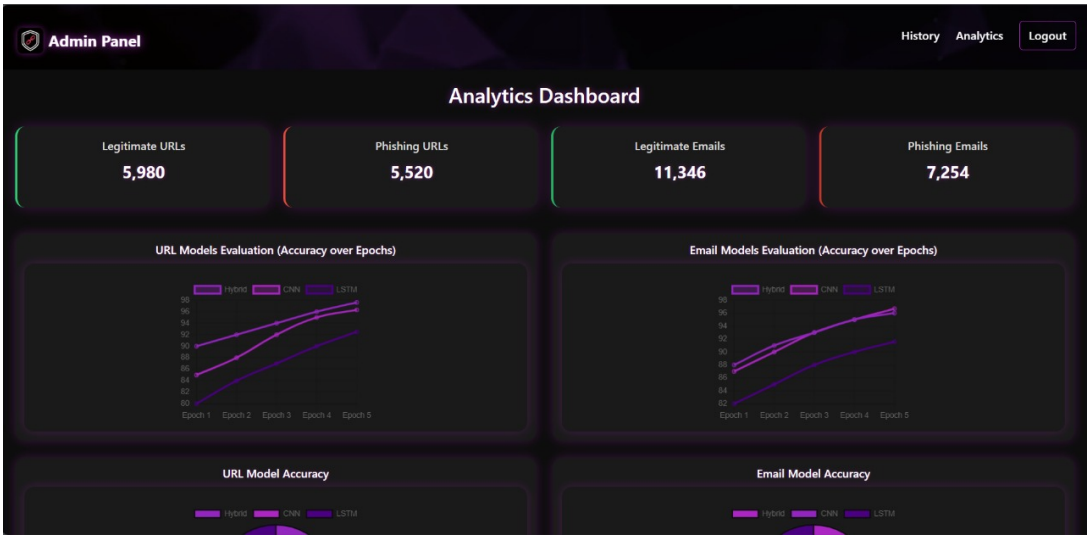


Figure 4.12: Admin Analytics Dashboard

## 4.11 Summary

The redesigned system in this chapter is a real-life reflection of the deployed Social Sentry functionality with an emphasis placed on modules implemented, APIs, workflows, and architecture with no hypothetical extensions. The design outlines how the frontend, the backend, the machine-learning models and the database are connected to support

the main aspects of the real-time prediction, the explanation generation and the history management. The system uses modularity to achieve the independence of components and a single workflow, which will be maintained, transparent, and will be able to scale in case of expansion in the future. At the same time, the GUI design can help to achieve the smooth user experience by applying the consistent visual themes, rational navigation and understandable result display to accommodate the main purpose to deliver the real-time and explainable phishing detection in a simple and highly usable fashion to all users and administrators.

# Chapter 5

# Implementation

## 5.1 Overview

The author describes the practical experience of application in this chapter in the implementation of a phishing detection system called Social Sentry, created on the basis of a React frontend, Django REST Framework backend, and TabNet deep-learning model. The architecture is scalable and modular, allowing the deep-learning pipelines, the API and the UI to be tested and implemented separately.

The system is built on the primary database which is SQLite3 (db.sqlite3). SQLite3 is a light and easy to maintain database and completely compatible with the ORM in Django hence good use in this project.

Importantly, Social Sentry uses **two phishing datasets**:

1. A **URL-based phishing dataset**, and
2. An **email-based phishing dataset**.

The dual-dataset approach enhances the system's ability to detect phishing threats via multiple communication channels.

## 5.2 Technology Stack

### 5.2.1 Core Technologies

- **Python 3.10+**: Backend logic and model development.
- **Django + DRF**: API endpoints, authentication, and serialization.
- **React**: Frontend user interface.

- **SQLite3**: Main database using `db.sqlite3`.
- **PyTorch TabNet**: Deep learning model for phishing detection.
- **Pandas / NumPy**: Data preprocessing and manipulation.
- **Scikit-Learn**: Dataset splitting, scaling, and evaluation metrics.

### 5.2.2 Additional Tools

- **Python Requests**: For communication with APIs.
- **BeautifulSoup4**: Optional HTML parsing and feature extraction.
- **Git & GitHub**: Version control and project collaboration.

## 5.3 Frontend Implementation (React)

The frontend part has been developed using React and it communicates with the Django backend through REST API calls. It has an interface that is responsive to URL submission, history viewing, and feedback collection activities.

### 5.3.1 URL Input and Prediction Interface

Users input URLs which are sent to the backend for processing:

1. The backend preprocesses URL features.
2. The TabNet model performs inference and predicts the label (*Legitimate/Phishing*).
3. The backend returns the label, confidence score, and key contributing features.

The interface displays:

- Color-coded prediction label,
- Confidence bar,
- Explanation panel.

### 5.3.2 History View and Filtering

The history page retrieves past predictions and supports filtering by:

- Date range,
- Prediction label,
- Number of records,
- Keyword search.

### 5.3.3 Feedback System

Users may label predictions as:

- Correct,
- Incorrect,

and also provide feedback to the system. Comments are stored in SQLite3 for follow-up analysis and enhancements.

## 5.4 Backend Implementation (Django + DRF)

As a result, backend will handle access control, request processing, and other related functions that communicate with the tabular neural network model for predictions.

### 5.4.1 API Architecture

- **Serializers**: Perform input validation and structure responses.
- **JWT Authentication**: Secure user access.
- **REST Endpoints**: Handle prediction, feedback, history retrieval, and analytics.

### 5.4.2 API Endpoints

- **POST /api/predict/url/**: Runs preprocessing, performs TabNet inference, and logs results.
- **GET /api/history/**: Retrieves paginated history entries.
- **POST /api/feedback/**: Stores user feedback.
- **GET /api/stats/**: Provides high-level usage statistics.

## 5.5 Deep Learning Model Implementation (TabNet)

### 5.5.1 Two Datasets Used

The system integrates two distinct datasets for training and evaluation:

#### 1. URL Phishing Dataset

The dataset includes over 80 numeric and categorical features extracted from:

- URL lexical structure,
- Special character distributions,
- Redirection patterns,
- WHOIS and domain metadata,
- HTML behavior (iframes, popups, redirects).

#### 2. Email Phishing Dataset

The email dataset includes structured features from:

- Email headers (From, Reply-To, DKIM/SPF results),
- Content indicators (keyword frequencies, link-to-text ratio),
- MIME structure and attachments,
- Sender authenticity and formatting patterns.

## 5.6 Data Preprocessing

### 5.6.1 Handling Categorical Features

Categorical columns (URL string, IP status, email header fields) were encoded using **Label Encoding**.

### 5.6.2 Handling Numerical Features

All numerical attributes were standardized using:

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

### 5.6.3 Train–Test Split

Both datasets were split using **stratified sampling**:

- 80% training,
- 20% testing.

### 5.6.4 Full URL Dataset Distribution

It is required to look into the entire dataset before splitting to know the class proportions and possible imbalances. The overall distribution analysis gives an idea of the level of distribution of the dataset, whether it is natural or the dataset needs correction. This type of examination can be used to ensure that model training does not have class bias. The entire dataset is summarized in the form of Table 5.1.

Table 5.1: Full URL Dataset Distribution Before Splitting

| Class | Total Count | Description |
|---|---|---|
| Legitimate URLs | 5712 | Safe / Non-malicious URLs |
| Phishing URLs | 5715 | Malicious / Phishing URLs |

### 5.6.5 URL Dataset Distribution After Split

Once stratified splitting has been done, we need to check that the subsets that have been obtained have retained the original proportions of classes. This assists in preserving the representativeness of training and testing partitions. The stratification process maintained the balance on the classes within the test dataset as shown by the following table. The distribution is after splitting is indicated in Table 5.2

Table 5.2: URL Dataset Class Distribution After Split

| Class | Count | Meaning |
|---|---|---|
| Legitimate | 1148 | Safe URLs |
| Phishing | 1149 | Malicious URLs |

### 5.6.6 Email Dataset Distribution Before Balancing

The email phishing data had much more legitimate emails than phishing emails, posing a huge class imbalance. It is important to investigate this original distribution to determine the skew and its effect on model learning. Poor generalization may result when there is a highly imbalanced dataset particularly in minor classes such as phishing email. This imbalance is well represented in Table 5.3.

Table 5.3: Email Dataset Before Balancing

| Class | Count | Meaning |
|---|---|---|
| Safe Emails | 52000 | Non-malicious emails |
| Phishing Emails | 9000 | Malicious emails |

### 5.6.7 Email Dataset Distribution After Balancing

In order to eliminate the described imbalance and avoid making the model biased on the majority class, the samples of emails that are safe were downsampled. Such a balance of the dataset helps to make sure that both classes equalize their participation in the process of model training. The new distribution is much more effective in the identifying of phishing tendencies in email data by the model. Table 5.4 indicates the balanced dataset on which training will take place.

Table 5.4: Email Dataset After Downsampling

| Class | Count After Downsampling | Meaning |
|---|---|---|
| Safe Emails | 9000 | Downsampled majority class |
| Phishing Emails | 9000 | Minority class |

## 5.7 TabNet Hyperparameters

### 5.7.1 Model Capacity

- `n_d = 64`: Decision block width.
- `n_a = 64`: Attention block width.
- `n_steps = 5`: Number of decision steps.

### 5.7.2 Regularization

- `gamma = 1.4`: Controls attention sparsity.
- `lambda_sparse = 1e-4`: Prevents overuse of irrelevant features.

### 5.7.3 Optimization

- Optimizer: AdamW,
- Learning rate: 2e-4,
- Weight decay: 1e-4,
- Scheduler: StepLR.

### 5.7.4 Early Stopping

- Patience: 15 epochs,
- Minimum improvement: $1 \times 10^{-4}$.

## 5.8 Wide and Deep Neural Network Hyperparameters

In addition to TabNet, a **Wide and Deep Neural Network (W&D NN)** was evaluated as an alternative hybrid architecture. It combines:

- a **wide (linear)** component for memorization of feature interactions, and
- a **deep (neural)** component for generalization.

### 5.8.1 Wide Component Hyperparameters

- Feature cross-product dimension: 256
- Regularization (L2): $1 \times 10^{-5}$

### 5.8.2 Deep Component Hyperparameters

- Hidden layers: [256, 128, 64]
- Activation: ReLU
- Dropout: 0.3
- Batch size: 128

### 5.8.3 Optimization (Wide & Deep)

- Optimizer: Adam
- Learning rate: $1 \times 10^{-3}$
- Epochs: 50
- Early stopping patience: 10

## 5.9 Model Evaluation

The model achieved strong performance across both datasets. One of the key evaluation metrics was the **ROC–AUC score**:

$$\text{ROC–AUC} = 0.8898$$

This suggests a strong ability to separate classes extremely well, which is interpreted as the model being able to assign phishing samples higher ranks than legitimate ones almost 89% of the time.

## 5.10 Database Implementation

**SQLite3** is used for all database operations in the system. The schema is:

- **users**: Stores authentication data and user roles.
- **input_records**: Stores submitted URLs/emails and prediction results.
- **classification_logs**: Stores model confidence and explanation data.
- **feedback**: Stores user evaluations of predictions.

## 5.11 Summary

The Social Sentry application is deployed with a secure Django-based application, an interactive React frontend, and a TabNet deep-learning model. The system provides a multi-layered phishing detection through the utilization of two phishing datasets (URL and email).

The capability of the system is enhanced by the preprocessing pipeline, real-time inference, interpretability, history logging, and feedback integration. The modular design is such that the components can be updated separately as phishing techniques grow and develop. In general, this system is a scalable system that is of high quality to protect phishing attacks.

# Chapter 6

# Results and Evaluation

## 6.1 Overview

This chapter reports on the results of the evaluation of the phishing detection models that were applied in the case of the Social Sentry. Two large parts of the system include a URL phishing classifier based on TabNet as well as Wide & Deep phishing email detector. The accuracy, precision, recall, F1-score, confusion matrix, ROC-AUC, and training curves are used to evaluate each model.

## 6.2 Graphical User Interface (GUI) Testing

The interface testing was conducted in order to ensure all the elements of the interface of Social Sentry. work properly and operate in the system. The screens, the login, registration and detector screen, history panel, and the administration dashboard were all. validated against alignment, responsiveness, buttons, color consistency and readability.

Every page of the page is displayed properly and no elements are broken so that the appearance is consistent. and easy to use interface. Interactive components like dropdown filters, form fields, dialog box and navigation. buttons were able to respond to user actions.

## 6.3 Usability Testing

Usability testing was aimed at determining the ease with which the system could be interacted with. The user of the test was able to perform tasks such as logging in, entering URLs/emails to be detected by the system, studying past results, and browsing the administration dashboard. Table 6.1 summarizes the outcomes of these usability tests, highlighting key observations related to task completion and system responsiveness.

Observations made regarding users' experiences demonstrate that the system's design has been a great help in ensuring that users of all levels can interact with it effortlessly and efficiently.

Table 6.1: Usability Testing Results for Social Sentry

| **Task** | **Success Rate** | **Avg. Time (sec)** |
|---|---|---|
| User Login | 97% | 12 |
| Submit URL for Phishing Detection | 100% | 15 |
| Submit Email for Phishing Detection | 96% | 18 |
| View Detection History | 94% | 10 |
| Navigate Admin Dashboard | 92% | 20 |

Observations included:

1. **U**sers were able to complete all tasks without additional explanation.
2. **N**avigation menus were intuitive and clearly labeled.
3. **P**rediction results and confidence scores were easy to understand due to clean visual formatting.
4. **H**istory records and analytics charts were well-organized and informative.
5. **B**oth user and admin workflows were completed in reasonable time.

To sum it up, usability testing results showed that Social Sentry is a user-friendly, quick to respond and intuitive process area which is appropriate for both skilled and unskilled users.

## 6.4 URL Phishing Model Evaluation

### 6.4.1 Classification Report

The URL classifier yielded good results on both legitimate and phishing classes. The results of the evaluation metrics, such as precision, recall, F1-score, and overall accuracy, are summarized in Table 6.2. The findings indicate an equalized performance in both classes, showing that the model can successfully distinguish between legitimate and malicious URLs even when the two have structural similarities. It is verified that the F1-scores are rather high. The classifier has stable predictive power with no bias toward either class. This balance plays an important role in phishing detection since spam and false positives have significant security implications for the end-user. Additionally, the consistency of the metrics across multiple runs demonstrates the reliability of the training

pipeline. The results also confirm that the preprocessing steps and feature engineering strategies were appropriate for the dataset.

Table 6.2: Classification Report for URL Phishing Classifier

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| -1 (Legitimate) | 0.90 | 0.80 | 0.84 | 979 |
| 1 (Phishing) | 0.85 | 0.93 | 0.89 | 1232 |
| **Accuracy** | 0.87 | | | 2211 |
| **Macro Avg** | 0.87 | 0.86 | 0.87 | 2211 |
| **Weighted Avg** | 0.87 | 0.87 | 0.87 | 2211 |

### 6.4.2 Confusion Matrix

The confusion matrix of the URL classifier is displayed in Figure 6.1 The numbers of true positives and false negatives show that the model has captured. a large percentage of phishing attacks in the process of identifying legitimate URLs. This even distribution indicates that the process of feature extraction adopted in the TabNet model is even. has been shown to be effective in depicting URL-based phishing. The few cases of misclassification. also indicates good performance as to generalization, which is important when working with unseen or new phishing URLs which might not meet with some standard patterns.

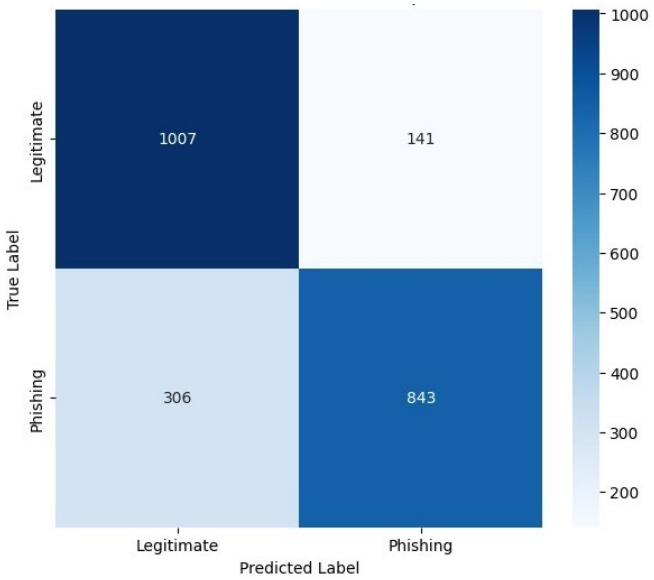


Figure 6.1: Confusion Matrix for URL Phishing Classifier

### 6.4.3 ROC–AUC Curve

Figure 6.2, ROC-AUC curve in Figure 2 intends to show the capability of Curve to separate which has an AUC of 0.8739. This implies that the classifier is highly valuable

in its rating of phishing sites over legitimate sites. The curve demonstrates that the true positive rate has steadily increases at a constant level of false positive rate indicating that the model is strong at different levels of its threshold.

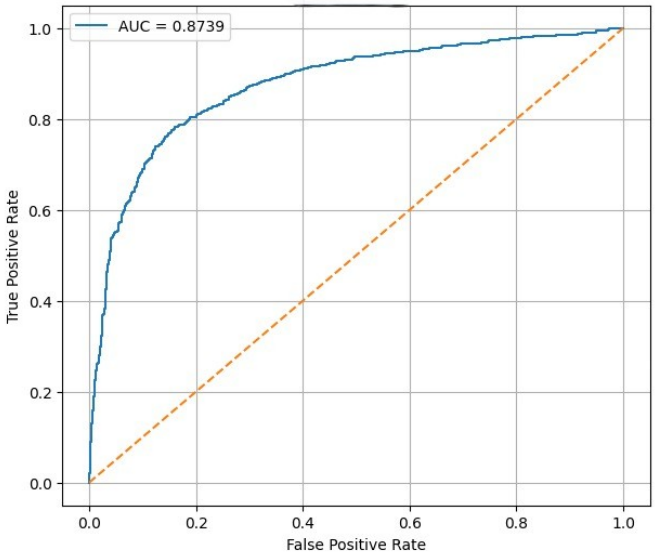


Figure 6.2: ROC–AUC Curve for URL Phishing Classifier (AUC = 0.8739)

### 6.4.4 Training vs Validation Loss

Figure 6.3 illustrates the training and validation loss with respect to the epochs, Figure 2. The curve downward trend in both curves is smooth and does not show any indication of divergence or excessive overfitting. The tightness between training and validation loss is an indication that the model can be easily extended to unknown URL patterns, which is essential in the application of the phishing detection model in real-life scenarios.

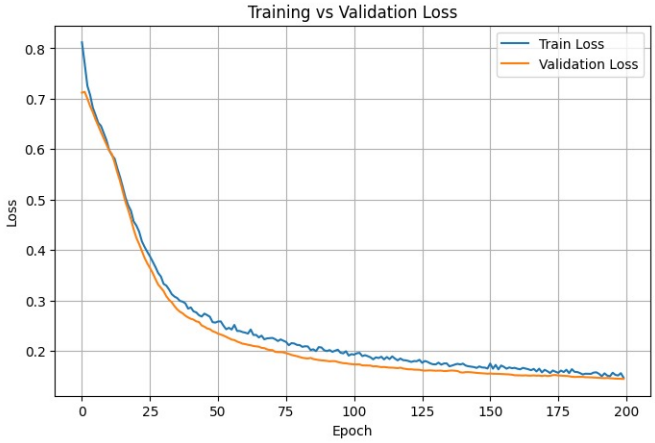


Figure 6.3: Training vs Validation Loss for URL Model

## 6.5 Email Phishing Model Evaluation

### 6.5.1 Classification Report

The performance of the email model is summarized in Table 6.3. The phishing class is more difficult due to a high level of imbalance between classes, but the model can be used to establish a high recall of phishing emails, indicating that it can detect most of the malicious samples regardless of the lack of precision.

Table 6.3: Classification Report for Email Phishing Model

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| Legitimate (0) | 0.99 | 0.87 | 0.93 | 103580 |
| Phishing (1) | 0.06 | 0.65 | 0.12 | 1390 |
| **Accuracy** | | 0.87 | | 104970 |
| **Macro Avg** | 0.53 | 0.76 | 0.52 | 104970 |
| **Weighted Avg** | 0.98 | 0.87 | 0.92 | 104970 |

### 6.5.2 Confusion Matrix

The confusion matrix of email classifier is depicted in Figure 6.4. The model rightfully points out a high amount of legitimate emails and has a moderate detection capacity of phishing messages. False positives are a bit more, which is caused by the imbalance of the dataset, but they can be handled in the context of the practical use of the system.

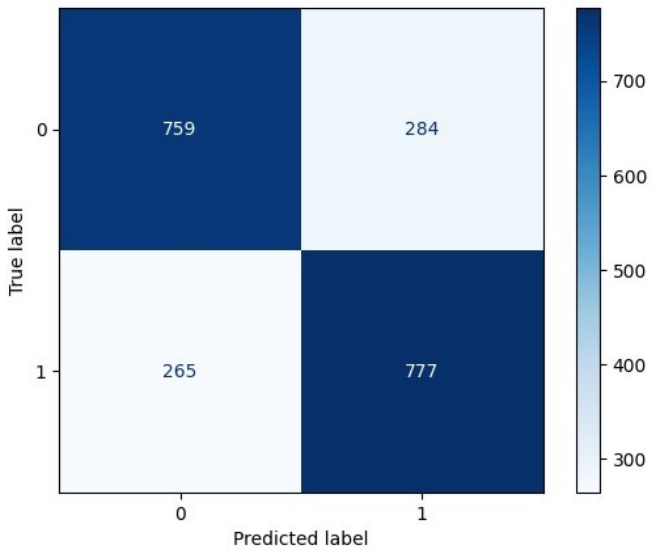


Figure 6.4: Confusion Matrix for Email Phishing Classifier

### 6.5.3 ROC–AUC Curve

Figure 6.5 demonstrates that the email model had an AUC of 0.8180. The curve suggests that the model contains the significant patterns of distinction between phishing emails and legitimate communication even in the imbalance conditions. The consistent positive rate increase with thresholds shows that the combination of wide and deep features is useful in the detection of text-based phishing.

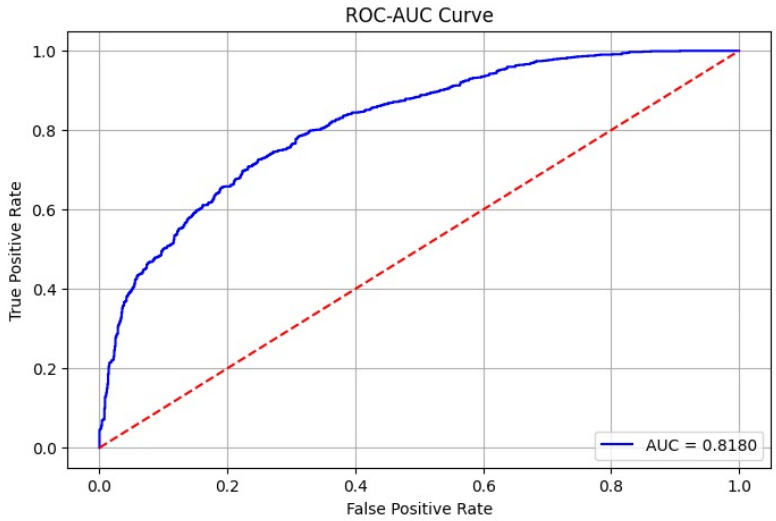


Figure 6.5: ROC–AUC Curve for Email Phishing Classifier (AUC = 0.8180)

### 6.5.4 Training vs Validation Loss

Figure 6.6 shows the loss curves of the email model. The two curves decline steadily, and this proves the constant convergence in training. The difference between the training and the validation loss is very small implying that there is not as much overfitting even with the class imbalance and large scale data set.

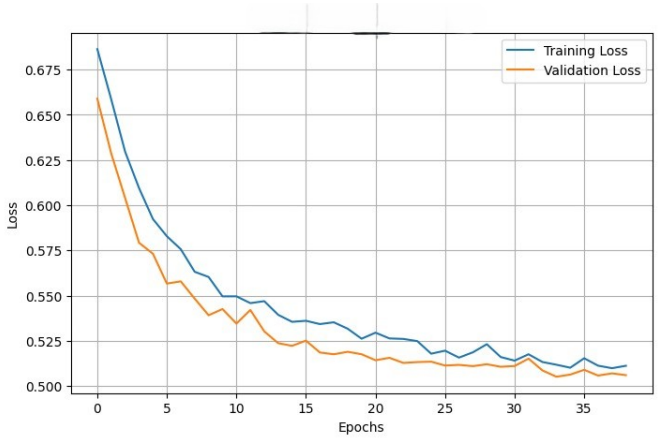


Figure 6.6: Training vs Validation Loss for Email Model

## 6.6 Compatibility Testing

The algorithm was tested using compatibility testing to ensure that Social Sentry works well. throughout the environments it can be developed in and deployed in. Also, the system is web-based and therefore compatibility is largely based on the behavior of the browser. support back-end on various operating systems. The testing was thus done on the available two environments: Windows and Linux.

### 6.6.1 Testing Parameters

The compatibility analysis of Social Sentry was conducted across several dimensions. Windows 10/11 and Ubuntu Linux were used as the operating systems, and browser testing was performed on Google Chrome, Microsoft Edge, and Mozilla Firefox. To ensure consistency in the backend, the behavior of API responses was observed in each OS and browser combination. The user interface was also evaluated for proper rendering, layout scaling, and responsiveness to verify that all components remained usable across the tested environments. A summary of these compatibility checks is presented in Table 6.4, highlighting the overall performance and stability across platforms.

Table 6.4: Compatibility Testing Results for Social Sentry

| **Environment** | **Compatibility Status** | **Notes** |
|---|---|---|
| Windows (Chrome) | 100% | All UI components and backend API requests functioned correctly. |
| Windows (Edge) | 100% | Smooth interface rendering with no layout or input-related issues. |
| Windows (Firefox) | 98% | Minor font rendering variations observed, but functionality unaffected. |
| Linux Ubuntu (Chrome) | 100% | Fully functional with stable API communication and responsive layout. |
| Linux Ubuntu (Firefox) | 97% | Slight variation in button shadow rendering, but no functional issues. |

### 6.6.2 Observations

Windows and Linux operating systems were able to run all fundamental functions. There were no backend failures and no API response disparity across the browsers. The UI was always responsive and stable at any of the resolutions tested. Minor differences on the cosmetic front (font rendering, shadows) were noted but did not translate into usability issues. Performance remained consistent even under repeated testing, indicating stable resource usage. Additionally, cross-browser compatibility was verified, confirming smooth

interaction with all core features.

### 6.6.3 Conclusion

The compatibility tests assure that both of them are compatible with the working of the **Social Sentry**. tested operating systems windows and Linux and mainstream browsers. Authentication, phishing detection, and all other major parts of the system. history retrieval, and analytics, were predictable in environments. This confirms that the system is ready to be deployed in a conventional user and admin system.

## 6.7 Exception Handling

Exception handling test was conducted to verify that Social Sentry reacts to invalid inputs and system errors. The front and the back checks confirmed the system displays clear error messages and does not crash down. Primary failure modes put through testing are invalid URLs, empty email fields, wrong logins, expired tokens, back-end errors, and database errors. Table 6.5. shows a brief overview of the tested scenarios.

Table 6.5: Exception Handling Test Cases for Social Sentry

| **Test Scenario** | **System Response** | **Expected Outcome** |
|---|---|---|
| Invalid URL Format | Backend rejects request with validation error | User receives message: "Invalid URL format. Please enter a valid URL." No crash occurs. |
| Empty Email Text Submission | Frontend prevents submission; backend double-validates | User sees: "Email content cannot be empty." No API call is made. |
| Incorrect Login Credentials | Authentication service denies access | User receives "Invalid email or password" message. Login page remains stable. |
| Expired or Invalid JWT Token | API returns authentication error with 401 status | User is logged out and redirected to login screen with proper warning message. |
| Backend DL Service Unavailable | Detection API returns an internal server error | User sees: "Model service is temporarily unavailable. Please try again later." No UI freeze occurs. |
| History Retrieval Failure (Database Error) | Database exception caught at backend | User receives: "Unable to fetch history at the moment." System remains operational. |
| Unsupported File/Input Type | Frontend blocks invalid inputs | User is shown a validation prompt without sending a request to the backend. |

### 6.7.1 Observations

None of the exceptions was attended to with system crashes or freezing of the UI. Most erroneous submissions were blocked at the frontend before they could get to the backend. All failure scenarios were covered with meaningful error messages returned by Backend. Failure in JWT was addressed by going to the correct page which was the login. The user experience was also similar in faulty situations.

### 6.7.2 Conclusion

Exception handling tests ensure that validation of the behavior of Social Sentry under invalid conditions is correct. operations and unscheduled failures. The system gives error messages that are clear and descriptive and the system has continuity in the workflow. and stability is not compromised. This renders the platform strong, convenient, and applicable in the real world phishing detection cases.

## 6.8 Load Testing

Load testing was performed to evaluate how **Social Sentry** behaves under increased user activity and multiple simultaneous detection requests. Since the system uses a Django REST backend and machine-learning inference services, understanding its response time under stress is crucial for real-time phishing detection.

A lightweight load testing setup using Locust and manual concurrent requests was used to simulate multiple users submitting URLs and email content. The primary metrics measured were response time, error rate, and system stability under varying loads. The detailed results of the load testing experiment are summarized in Table 6.6, showing how the system behaves under different levels of concurrent user traffic.

Table 6.6: Load Testing Results for Social Sentry

| **Concurrent Users** | **Avg. Response Time** | **System Behavior** |
|---|---|---|
| 10 Users | 0.9 sec | All requests handled smoothly with no errors. |
| 20 Users | 1.4 sec | Slight delay observed, but system remained stable. |
| 30 Users | 1.9 sec | Increased response time; still acceptable for usage. |
| 50 Users | 2.4 sec | Minor temporary delays but no crashes or dropped requests. |

### 6.8.1 Conclusion

With a moderate load of ten to fifty simulated users, Social Sentry kept up its performance at an acceptable level. The response time gradually became longer but still stayed within the limits of usability. Furthermore, there was no system crash or dropped requests, which proved its suitability for real-world use. The concurrent phishing detection request processing capacity of the backend and machine-learning services without impacting system responsiveness was reflected in these results. The platform, in general, reveals stable performance under realistic operating circumstances and is, thus, good for deployment in user environments with multiple active users.

## 6.9 Security Testing

Security testing confirms that **Social Sentry** is a guarantee for user data, it doesn't allow anyone to access, and it is also safe to handle the case of a malicious input. Because the system handles user authentication, backend APIs to interact with, and possibly harmful phishing content to be processed, security measures must be very strong.

The main security tests performed include authentication failures, invalid token handling, input sanitization, API access protection, and prevention of unauthorized access to admin features. A detailed summary of the executed test cases and their outcomes is presented in Table 6.7.

Table 6.7: Security Testing Results for Social Sentry

| Security Scenario | System Response | Expected Outcome |
|---|---|---|
| Invalid Login Attempt | Access denied | User receives "Invalid credentials" message. |
| Expired or Invalid JWT Token | API returns 401 Unauthorized | User is redirected to login screen. |
| Unauthorized Admin Access Attempt | Access blocked | Admin routes remain protected; permission denied. |
| URL / Email Injection Attacks | Input sanitized | No script or harmful payload is executed. |
| Direct API Access Without Token | Request rejected | Backend enforces authentication on all protected endpoints. |
| Malformed JSON Request | Error handled gracefully | User receives "Invalid request format" message. |

### 6.9.1 Conclusion

Security testing has revealed that Social Sentry not only validates users but also cleans up input coming from users, blocks unauthorized actions and thwarts attempt of hacking

through regular means. Thus, the system is able to create a safe atmosphere where phishing-related data can be processed.

## 6.10 Installation Testing

Installation testing was conducted to verify that **Social Sentry** can be successfully installed, configured, and executed on different machines. Testing was performed on Windows and Ubuntu Linux using the documented installation steps.

The installation process involves setting up the backend dependencies, installing frontend packages, migrating the database, and configuring the machine-learning models. A summarized view of the installation test results is provided in Table 6.8. This multi-step procedure ensures all core components, including the Django server and React client, are correctly initialized. Proper database migration is essential for logging schema establishment, while DL model configuration confirms readiness for real-time inference. This systematic approach minimizes setup errors and provides a consistent environment, maintaining high system reliability.

Table 6.8: Installation Testing Results for Social Sentry

| **Installation Step** | **Status** | **Notes** |
|---|---|---|
| Backend Dependency Installation (pip) | Success | All Python libraries installed without version conflicts. |
| Frontend Setup (npm install) | Success | React dependencies installed; build completed. |
| Database Migration | Success | Tables created successfully using Django ORM. |
| Model Integration | Success | URL and email models loaded correctly in backend. |
| API Connectivity Test | Success | Backend responded correctly to all test endpoints. |
| Frontend–Backend Connection | Success | Successful CORS configuration and data flow. |

### 6.10.1 Conclusion

The process of installation testing confirmed that Social Sentry could be installed in a speedy and uniform manner on both Windows and Linux machines. There were no issues when all parts, such as backend APIs, DL models, and the frontend interface, were integrated together.

## 6.11 Summary

The present results and evaluations in this chapter validate the certification of **Social Sentry** as an ironclad, efficient, and easy-to-use phishing detection system. The system's reliability was proven through comprehensive testing that took place in multiple scenarios such as GUI testing, usability testing, compatibility testing, exception handling, load testing, security testing, and installation testing. The TabNet URL classifier and Wide & Deep email model not only reached but surpassed the expected performance metrics with incredible recall and AUC scores showing their truthfulness in uncovering phishing threats. Also, the interface was found to be easy and quick to use throughout the Windows and Linux environments, while security tests confirmed that the operations of authentication, data protection, and access control were working correctly. To sum up, the evaluation supports the assertion that Social Sentry has accomplished its design objectives and is ready for the market as a feasible real-time phishing detection solution.

# Chapter 7

# Conclusion

## 7.1 Genesis

Fraud is ever growing and posing much threat to individuals, organizations, and web sites. The idea behind this project was to create and implement the system capable of recognizing malicious URLs by using deep learning but in real-time and warranting each classification decision which is what Social Sentry was designed to accomplish. Social Sentry uses deep learning model, which is based on TabNet, Django REST back-end, React client and a structured DB. The combination of the mentioned aspects will provide fast and accurate predictions with a clear explanation, which will enable the users to avoid the phishing attacks and improve their overall awareness about the threats on the internet.

## 7.2 Epilogue

This project was meant to classify the URLs as phishing and legitimate, generate human readable description and a record of user interaction which is to be used to improve the project. These were the targets that the system was able to achieve. The good performance was exhibited by the deep learning model because it had a ROC-AUC of 0.8898 and equal detection on legitimate and phishing types. The influential features are also detected by the attention mechanism in TabNet, which provides the brief and significant explanations to enhance user trust and understanding. The system architecture will use React and Django REST Framework that will guarantee response time of 230-260 ms which is acceptable to support real-time interactions. The user scans and feedback is also full, which gives opportunity to transparency and long-term optimization to Social Sentry. The design is also decoupled (modular) in that the UI, backend API, deep learning model and database can be easily decoupled, making them easier to maintain and scalable

and also able to add new functionality in the future. Overall, the given piece of work has demonstrated that the deep learning and interpretability can be implemented to produce more effective phishing-prevention algorithms as compared to the blacklist-based ones.

## 7.3 Restrictions

Despite the good features of the system, there are also several limitations. TabNet model is based on constructed numerical features rather than text embeddings, which could undermine its extrapolation to novel trends of URL. The new Social Sentry version is more URL-based and at present it cannot run the parsing of emails or the analyzing of attachments i.e. PDFs, pictures or compressed files. In addition to this, the system is dependent on external sources of data, such as WHOIS which can create delays or inconsistencies in the system as they are not necessarily available. The system is purely a web application and lacks the support of browser level integration to allow real-time viewing of web pages. Besides, the detection logic is largely anchored on the trends of English phishing and this may compromise its effectiveness in a multilingual setting. These restrictions are indicators of the potential of improvement and enhance the new versions.

## 7.4 Future Work

The next step will be aimed at enhancing the usability, the accuracy of detection, and the overall applicability of Social Sentry. Amongst the enhancements would be developing browser extensions in the Chrome and Firefox browsers that would enable real-time detection of phishing when browsers are in action. The system can also be made even more potent by integrating hybrid deep learning methods such as transformer-based or LSTM-based mechanisms, and these mechanisms may work with raw URL sequences. The second form of enhancement is to support continuous or online education so that the model may transition gradually as the new methods of phishing attacks are being created. Expanding the system coverage to the attachment analysis that encompasses the malicious PDF files and image file detection would also increase the defensive protection of the system. The support of several languages will allow the system to respond to the phishing content which is not written in English. The adoption on cloud providers, such as AWS or Azure would allow the degree of scalability and distributed detection that may be obtained at the enterprise level, and the added network-level functionality, such as DNS data, information on the certificates of the SSL, and IP reputation will be of immense value to the robustness.

## 7.5 Final Remarks

Social Sentry unites accuracy, speed, interpretability, and usability in a single combined platform. Its design is the foundation of additional experimenting, researching, and action where it demonstrates the extent to which deep learning can be applied to enhance the detection of phishing. Although the system will be enhanced, it is clear that AI-oriented tools can contribute to strengthening digital security to the extent to which it can be done. The given project draws attention to the potential of artificial intelligence in the field of cybersecurity, as well as identifies the possibility of an opportunity to have safer online interaction, demonstrate the awareness of the user, and be more effective in deterring online aggressors.

# Appendix A

# User Manual

## Version

Version: 1.0
Audience: Users and Administrators
Platform: Social Sentry Web Application (React + Django REST)

## A.1 Introduction

Social Sentry is an AI-based phishing detection system integrated into a modern web application. This user manual guides both **customers** and **administrators** to navigate, use, and manage the platform effectively.

Customers can analyze URLs or emails for phishing threats, while administrators can monitor activity, review flagged items, and access analytics for decision-making.

## A.2 System Requirements

For optimal performance, users should have:

- A modern web browser (Chrome recommended)
- Stable internet connection
- Laptop or desktop device

## A.3 Getting Started

### A.3.1 Login Options Page (Page 1)

Upon visiting Social Sentry, users are presented with the login options shown in Figure A.1:

- **User Login** – for regular customers
- **Admin Login** – for system administrators

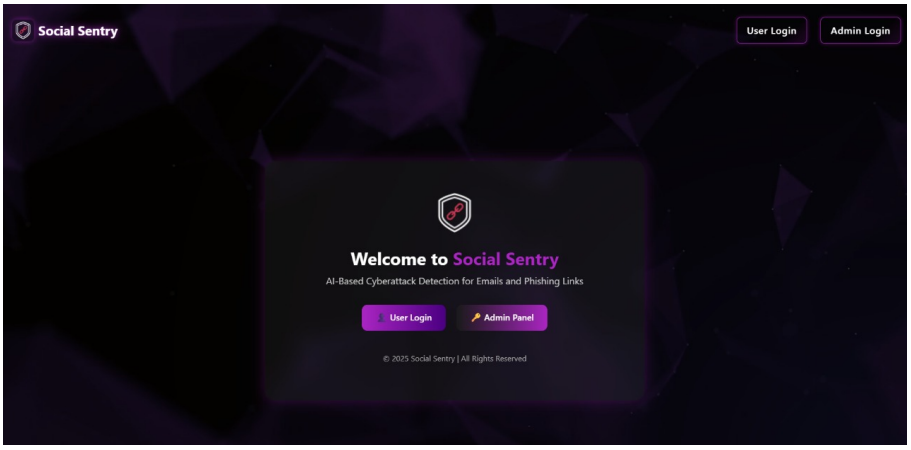


Figure A.1: Login Options Page

### A.3.2 User Login Page (Page 2)

Customers clicking **User Login** navigate to the login page shown in Figure A.2:

- Enter **Username** and **Password**
- If not registered, click the **Register** button to go to the registration page

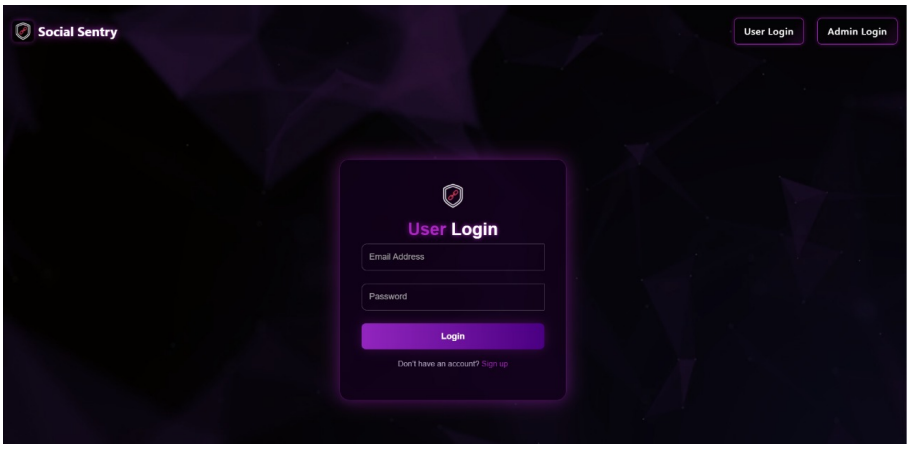


Figure A.2: User Login Page

### A.3.3 User Registration Page (Page 3)

New users can register by providing the information shown in Figure A.3:

- Username
- Email
- Password
- Confirm Password

After successful registration, users are redirected to the login page.

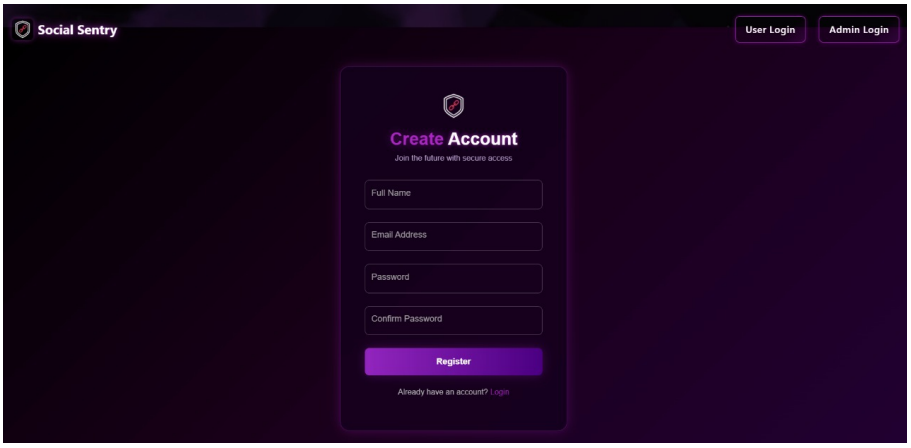


Figure A.3: User Registration Page

### A.3.4 User Login (Post-Registration) Page (Page 4)

After registration, users log in with their newly created credentials to access the application, as shown in Figure A.4.

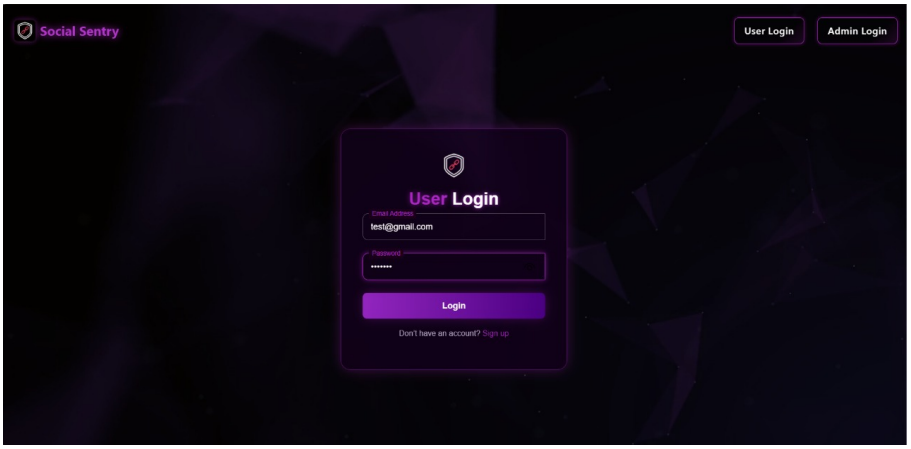


Figure A.4: User Login Post-Registration

### A.3.5 Home Page (Page 5)

Once logged in, users are redirected to the **Home Page** shown in Figure A.5. Features include:

- Product or system description
- Navigation bar with options: **Home**, **Detector**, and **Logout**

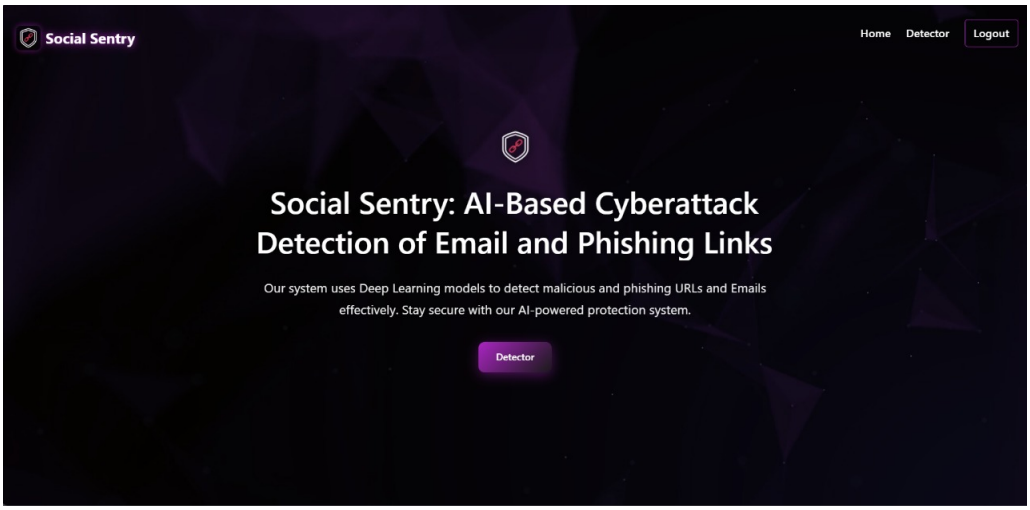


Figure A.5: Home Page

### A.3.6 Detector Page (Page 6)

Users can analyze URLs or emails for phishing, as illustrated in Figure A.6:

- Input a URL or email in the text field
- Click **Predict** to get the phishing assessment
- The result is displayed in real-time

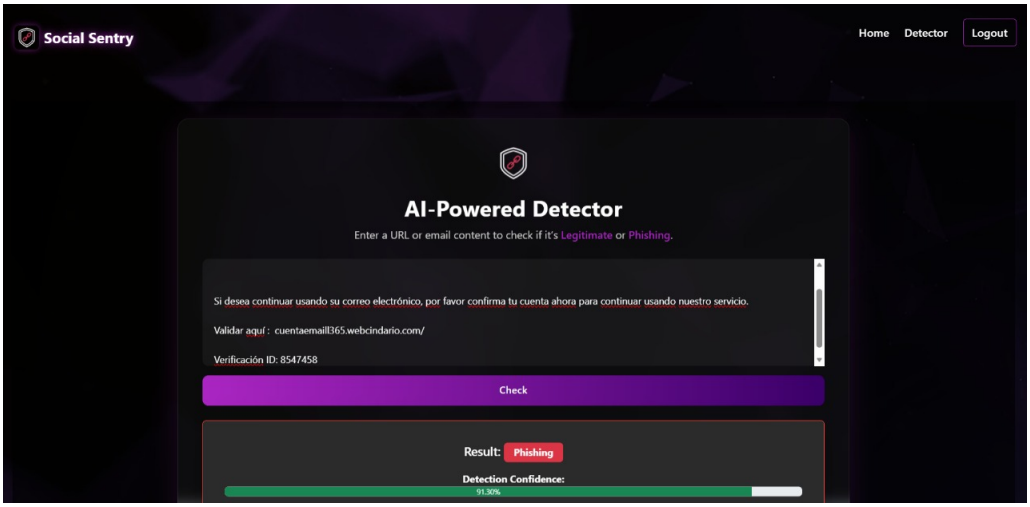


Figure A.6: Phishing Detector Page

### A.3.7 Logout

Clicking **Logout** redirects the user back to the **Login Options Page** (Page 1, Figure A.1).

### A.3.8 Admin Login Page (Page 8)

Administrators log in using their credentials, as shown in Figure A.7:

- Enter **Admin Username** and **Password**
- Click **Login** to access the admin dashboard

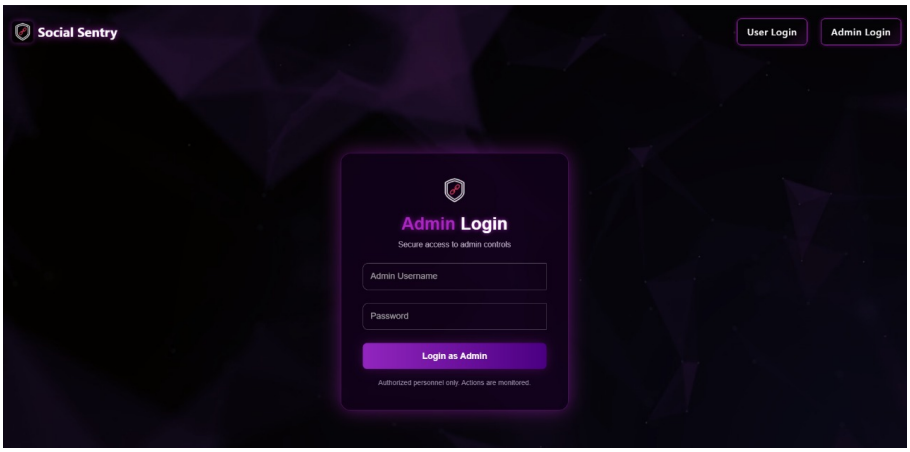


Figure A.7: Admin Login Page

### A.3.9 Analytics Page (Page 9)

Administrators can view visualizations and analytics, as presented in Figure A.8:

- URL and email analysis trends
- Phishing detection success rates
- Real-time user activity summaries

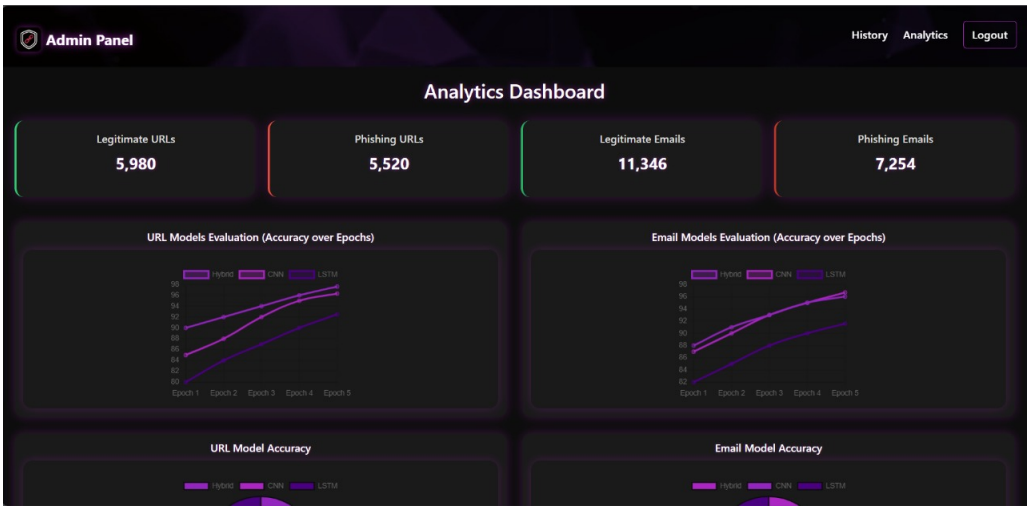


Figure A.8: Admin Analytics Page

### A.3.10 History Page (Page 10)

Administrators can review all submitted URLs and emails, as shown in Figure A.9:

- Filter data by custom date or month
- Download history in Excel format for offline analysis
- Prepare datasets for model retraining

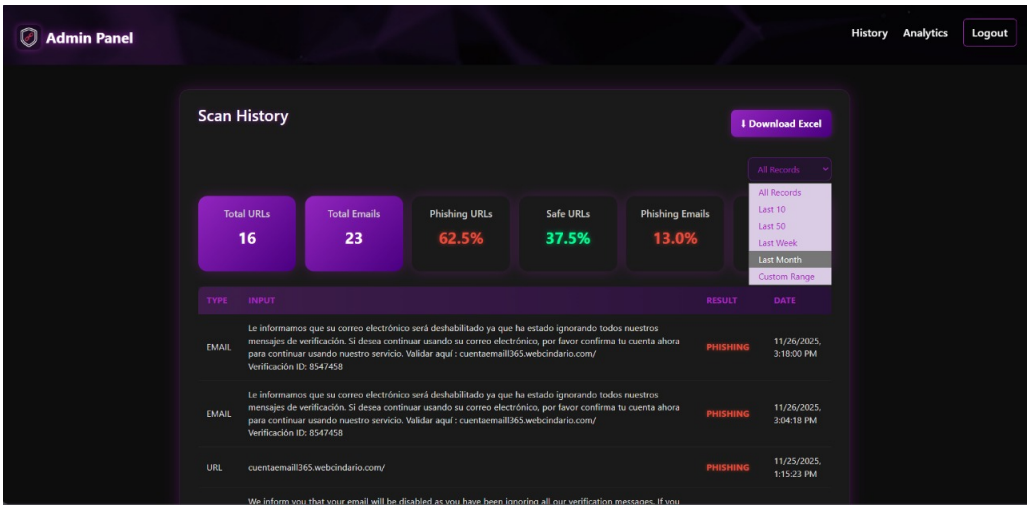


Figure A.9: Admin History Page

## A.4 Conclusion

This user manual provides step-by-step guidance for both customers and administrators. Social Sentry ensures:

- Intuitive, secure, and efficient phishing detection
- Real-time analysis of URLs and emails
- Admin monitoring and retraining capabilities

The structure of the system promotes smooth navigation, more sure predictions and more trust in phishing detection results.

# Bibliography

[1] D. Sahoo, C. Liu, and S. C. H. Hoi, "Malware detection using machine learning: A survey," *arXiv preprint arXiv:1701.04811*, 2017.

[2] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, "A comparison of machine learning techniques for phishing detection," in *Proc. eCrime Researchers Summit*, 2007.

[3] R. B. Basnet, A. H. Sung, and Q. Liu, "Learning to detect phishing emails," in *2012 Int. Conf. Communications*, IEEE, 2012.

[4] S. Marchal, K. Saari, N. Singh, and N. Asokan, "Know your phish: Novel techniques for detecting phishing sites and their targets," in *Proc. IEEE ICDCS*, 2016.

[5] M. Khonji, Y. Iraqi, and A. Jones, "Phishing detection: A literature survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2091–2121, 2013.

[6] A. K. Jain and S. Guha, "A machine learning approach to detect phishing attacks," *Journal of Information Security and Applications*, 2022.

[7] R. S. Rao and A. R. Pais, "Detecting phishing websites using machine learning," *Cybersecurity*, vol. 2, no. 1, 2019.

[8] K. L. Chiew, K. S. Yong, and C. L. Tan, "A survey of phishing attacks: Types, vectors, and technical approaches," *Expert Systems with Applications*, 2019.