A PROJECT REPORT ON

PHISHING URL CLASSIFICATION USING MACHINE LEARNING

Submitted in the partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY HONORS

In

Computer Science and Engineering

By

Shaik Subhan Saheb (20A81A05I3)

Under Esteemed Supervision of

Mr. M.S. Kumar Reddy, MTech, (Ph.D), Asst.Professor

Department of Computer Science and Engineering (Accredited by N.B.A.)

SRI VASAVI ENGINEERING COLLEGE (Autonomous)

Pedatadepalli, Tadepalligudem-534101, A.P

2023-24

SRI VASAVI ENGINEERING COLLEGE (Autonomous)

Department Of Computer Science and Engineering

Pedatadepalli, Tadepalligudem

Certificate

This is to certify that the Project Report entitled "**Phishing URL Classification Using Machine Learning**" submitted by Shaik Subhan Saheb (20A81A05I3) for the award of the degree of Bachelor of Technology Honors in the Department of Computer Science and Engineering during the academic year 2023-2024.

Project Guide

Mr. M. S. Kumar Reddy MTech, (Ph.D)

Assistant Professor

Head of The Department

Dr.D.Jaya Kumari M.Tech, Ph.D

Professor & HOD

External Examiner

DECLARATION

I hereby declare that the project report entitled "**Phishing URL Classification Using Machine Learning**" submitted by me to Sri Vasavi Engineering College (Autonomous), Tadepalligudem, affiliated to JNTUK Kakinada in partial fulfilment of the requirement for the award of the degree of B. Tech Honors in Computer Science and Engineering is a record of Bonafide project work carried out by me under the guidance of Mr. M. S. Kumar Reddy, Assistant professor. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree in this institute or any other institute or University.

Project Associate

Shaik Subhan Saheb(20A81A05I3)

ACKNOWLEDGEMENT

First and foremost, I sincerely salute to our esteemed institute SRI VASAVI ENGINEERING COLLEGE, for giving me this golden opportunity to fulfil our warm dream to become an engineer.

My sincere gratitude to my project guide Mr. M. S. Kumar Reddy, Assistant professor, Department of Computer Science and Engineering, for his timely cooperation and valuable suggestions while carrying out this project.

I express my sincere thanks and heartful gratitude to Dr. D. Jaya Kumari, Professor & Head of the Department of Computer Science and Engineering, for permitting me to do our project.

I express my sincere thanks and heartful gratitude to Dr. G.V.N.S.R. Ratnakara Rao, Principal, for providing a favourable environment and supporting me during the development of this project.

My special thanks to the management and all the teaching and non-teaching staff members, Department of Computer Science and Engineering, for their support and cooperation in various ways during my project work. It is my pleasure to acknowledge the help of all those respected individuals.

I would like to express my gratitude to my parents, friends who helped to complete this project.

Project Associate

Shaik Subhan Saheb(20A81A05I3)

ABSTRACT

The exponential growth of digital world providing great opportunities for both positive advancements and malicious activities. While the internet facilitates seamless communication, collaboration, and innovation, it has also become a breeding ground for cyber threats, with phishing attacks being a prominent concern. Phishing, a deceptive practice where attackers masquerade as trustworthy entities to trick individuals into revealing sensitive information, poses a significant risk to online security.

In response to the ever-evolving landscape of cyber threats, this project introduces a cutting-edge approach to identify phishing attack URLs using machine learning. Leveraging the power of machine learning, the proposing Phishing URL Classification System aims to dynamically adapt to the latest tactics employed by malicious actors. By extracting and analysing a diverse range of features from URLs, including lexical, structural, and content-based attributes, the system enhances its ability to distinguish between legitimate and phishing URLs. I am working to train the Machine Learning model over 10,000 distinct URLs to make the system more accurate.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER – 1
INTRODUCTION

1.1. INTRODUCTION

In the realm of cybersecurity, one of the critical challenges faced by organizations and individuals is the identification and prevention of phishing attacks. Phishing attacks involve deceptive emails, messages, or websites that aim to trick users into revealing their sensitive information. These attacks can lead to significant financial losses, compromised personal data, and damage to reputations. To combat this growing threat, researchers and cybersecurity experts have turned to machine learning techniques for phishing URL classification.

Phishing URL classification involves the analysis of various URL features to identify potential threats. Key features that are considered in this analysis include the length of the domain, the number of subdomains, the use of special characters, and the presence of HTTPS. By training machine learning models on these URL features, it becomes possible to distinguish between legitimate URLs and phishing URLs, thereby enhancing cybersecurity efforts.

1.2. MOTIVATION

The motivation for employing URL features in phishing website detection lies in the critical need to fortify online security. Detecting phishing sites through URL analysis serves as a frontline defence, shielding users from scams and fraud while safeguarding sensitive financial and personal information. This approach not only protects individuals but also mitigates financial losses, preserves brand integrity, and ensures business continuity. The motivation for phishing website detection using URL features is anchored in the urgent necessity to combat the escalating threat landscape of online scams. Analysing URL features provides a multifaceted approach to fortifying cybersecurity, encompassing user protection, financial security, and data integrity.

1.3. SCOPE

The scope for phishing website detection using URL features is vast and pivotal in addressing the ever-evolving landscape of cyber threats. Leveraging URL characteristics provides a comprehensive framework to proactively identify and mitigate phishing attacks. This approach extends across multiple domains, including user protection, financial security, and data integrity. The

scope encompasses the development of advanced machine learning models trained on diverse datasets, enabling the detection of subtle patterns and variations in deceptive URLs. As phishing techniques continue to grow in sophistication, the scope for URL feature analysis widens, creating opportunities for continuous innovation in cybersecurity measures. From protecting individual users to fortifying the security infrastructure of organizations, the scope for leveraging URL features in phishing detection plays a crucial role in enhancing the resilience of digital ecosystems worldwide.

1.4. PROJECT OUTLINE

The main objective of the work is to predict the class of a URL based on features using ML algorithm by providing user friendly environment. This automated system is used to take the features and predict the class of the URL. The outline of this work could be described simply as, the system is an API which takes POST data with URL parameter. And then extract features of the URL to use Random Forest Machine Learning estimator and predict class of the URL.

CHAPTER – 2
LITERATURE SURVEY

LITERATURE SURVEY

[1] M. Zabihimayvan and D. Doran, "Fuzzy Rough Set Feature Selection to Enhance Phishing Attack Detection", 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), New Orleans, LA, USA, 2019, pp. 1-6, DOI: 10.1109/FUZZ-IEEE.2019.8858884.

In this study, they used Fuzzy Rough Set theory to identify key features for detecting phishing URLs. Their evaluation, conducted on three benchmarked datasets, showed that FRS-selected features outperformed other methods, achieving a maximum F-measure of 95% with Random Forest classification. This suggests that their approach enables rapid and robust phishing detection without relying on external inquiries, enhancing resilience against zero-day attacks.

[2] I. Kara, M. Ok and A. Ozaday, "Characteristics of Understanding URLs and Domain Names Features: The Detection of Phishing Websites with Machine Learning Methods", in IEEE Access, vol. 10, pp. 124420-124428, 2022, DOI: 10.1109/ACCESS.2022.3223111.

The study utilized internet URLs data to identify phishing attempts, employing the random forest model for analysis. Results on both trained and untrained data showed high prediction rates, indicating the model's effectiveness. Specifically, 6,118 phishing and 3,610 legitimate websites were successfully detected out of 9,879 records. The study also addressed the importance of considering local and global attack factors in information security approaches, highlighting the need for targeted defense strategies.

[3] S. Parekh, D. Parikh, S. Kotak and S. Sankhe, "A New Method for Detection of Phishing Websites: URL Detection", 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, 2018, pp. 949-952, DOI: 10.1109/ICICCT.2018.8473085.

This paper introduces a novel method for phishing website detection using random forests and RStudio. By evaluating 31 proposed features, it demonstrates their effectiveness in detecting phishing sites, achieving accuracy level of around 90% with random forests. Various performance metrics, including true positives and negatives, F-measure, ROC, precision, and sensitivity, were utilized for analysis, providing insights

into the detection process. The study acknowledges the evolving nature of phishing attacks and suggests the need for advanced browser capabilities to detect and prevent such threats. Future research aims to develop self-learning systems capable of identifying new phishing attack types.

[4] M. M. Vilas, K. P. Ghansham, S. P. Jaypralash and P. Shila, "Detection of Phishing Website Using Machine Learning Approach", 2019 4th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT), Mysuru, India, 2019, pp. 384-389, DOI: 10.1109/ICEECCOT46775.2019.9114695.

They have studied various phishing attacks on URLs. By employing Extreme Learning Machine, they identified phishing websites. When individuals visit any website, features are extracted from its URL. The resulting features serve as test data. The objective of this technique is to detect fake or illegal websites and notify users in advance to prevent the misuse of their private information.

[5] N. F. Abedin, R. Bawm, T. Sarwar, M. Saifuddin, M. A. Rahman and S. Hossain, "Phishing Attack Detection using Machine Learning Classification Techniques", 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), Thoothukudi, India, 2020, pp. 1125-1130, doi: 10.1109/ICISS49785.2020.9315895.

In this paper, the performance of three widely used machine learning classifiers are compared. Among these three classifiers, random forest performance is the highest. The AUC of the random forest is 1.0, which means our system can detect phishing website a high accuracy. In future, the accuracy improvement task will be done by changing features. Handling large data and efficient machine learning model-based systems can be developed detect a phishing attack from a logged dataset.

CHAPTER – 3
SYSTEM STUDY AND ANALYSIS

3.1. Problem Statement

Phishing attacks via URLs continue to pose a significant threat to cybersecurity. Existing methods for detecting phishing URLs are often insufficient due to the dynamic nature of these attacks and the large volume of websites to monitor. Manual inspection and rule-based approaches are labour-intensive and prone to errors. Therefore, there is a critical need for automated and accurate techniques to detect phishing URLs efficiently. This study aims to address this challenge by developing machine learning-based approaches for robust phishing URL classification, mitigating the risks associated with online fraud and data breaches.

3.2. Limitations of Existing System

While existing systems incorporate machine learning (ML) algorithms for phishing URL detection, they often lack real-time accessibility and integration capabilities with external systems. In response, this project aims to develop an API that provides instant URL analysis and suggestions for terminals and other systems, such as email senders. By extending the capabilities of current ML-based detection systems, this API will offer seamless integration, allowing for timely decision-making and proactive prevention of phishing threats across various platforms.

3.3. Proposed System

The proposed system aims to create a machine learning (ML) solution for identifying phishing URLs. With a dataset [6] comprising 10,000 records, the ML model will undergo comprehensive training to effectively differentiate between legitimate and malicious URLs. Once deployed across diverse digital platforms, the system will continuously analyse URLs in real-time, promptly alerting users to potential phishing risks. To maximize accessibility, the ML-based detection system will be packaged into an API, facilitating effortless integration into existing software frameworks and bolstering overall cybersecurity defences.

3.4. Advantages of Proposed System

The proposed system offers several advantages over existing approaches to phishing URL classification, enhancing cybersecurity and user experience. These advantages include:

- Enhanced Accuracy through machine learning techniques trained on a comprehensive dataset.
- Real-Time Analysis providing instant warnings about potential phishing threats.

- Seamless Integration into existing software solutions via an API.
- User-Friendly Interface accessible to users of varying technical backgrounds.
- Cost-Effective Solution reducing operational costs compared to manual methods.
- Scalability to handle large volumes of data and evolving phishing attacks.
- Enhanced Cybersecurity by safeguarding sensitive information and preventing unauthorized access.
- Improved User Experience instilling confidence in the safety of digital interactions.

3.5. Functional Requirements

Functional requirements are the functions or features that must be included in any system to satisfy the business needs and to be acceptable to the users based on this the functional requirements that the system must require are as follows:

- System should collect URL from user
- System should get features from URL and classifies its class
- System has to render the predicted output to user

3.6. Non-Functional Requirements

Non-Functional requirements are a description of features, characteristics and attribute of the system as well as any constraints that may limit the boundaries of the proposed system. The non-functional requirements are essentially based on the performance, information, economy, control and security efficiency and services. Based on these the non-functional requirements are as follows:

- User friendly
- System should provide better accuracy
- To perform with efficient response time
- Availability
- Scalability

3.7. System Requirements

3.7.1. Software Requirements

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the

software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Server-Side Requirements:

Operating System	Windows 8/10
Coding Language	Python3
Framework	Django

Client-Side Requirements:

Active internet connected Browser/Terminal

3.7.2. Hardware Requirements

The hardware requirements are the requirements of a hardware device. Most hardware only has operating system requirements or compatibility.

Server-Side Requirements:

Processor Support	i3 onwards
RAM	4GB

Client-Side Requirements:

Active internet connected Browser/Terminal

CHAPTER – 4
SYSTEM ARCHITECTURE

4.1. SYSTEM ARCHITECTURE DESIGN

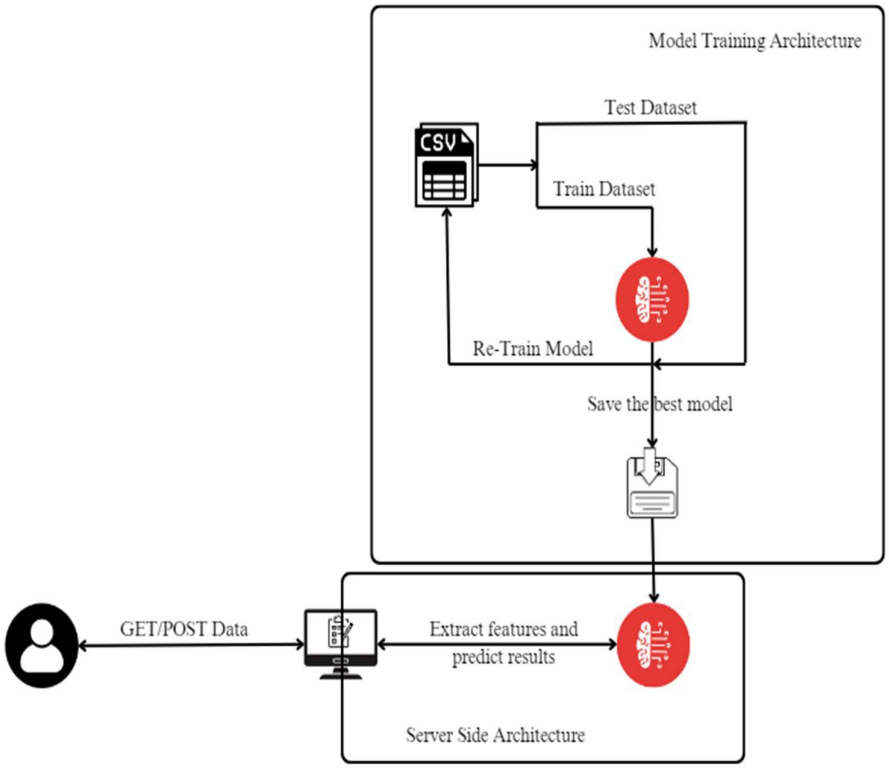

Figure 1 : System Architecture

The dataset of 10,000 records is trained on a machine learning algorithm several times to get better and accurate estimator. After getting a best estimator, that will be saved into memory.

User provides the URL data to serve either from a frontend web interface or terminal. Then important features are extracted from the URL. The machine learning model is loaded from memory and the extracted features are passed to the loaded model. Then the predicted class label is render back to user.

4.2. UML DIAGRAMS

A UML diagram shows the unified visual presentation of the UML (Unified Modelling Language) system intending to let developers or business owners understand, analyze, and undertake the structure and behaviours of their system. So far, the UML diagram has become one of the most common business process modelling tools, which is also highly significant to the development of object-oriented software.

UML diagrams have many benefits for both software developers and businesspeople, and the most key advantages are:

- **Problem-Solving** - Enterprises can improve their product quality and reduce cost especially for complex systems in large scale. Some other real-life problems including physical distribution or security can be solved.
- **Improve Productivity** - By using the UML diagram, everyone in the team is on the same page and lots of time is saved down the line.
- **Easy to Understand** - Since different roles are interested in different aspects of the system, the UML diagram offers non-professional developers, for example, stakeholders, designers, or business researchers, a clear and expressive presentation of requirements, functions and processes of their system.

USE CASE DIAGRAM:

A use case diagram is the primary form of system/software requirements for a new software program underdeveloped. Use cases specify the expected behaviour, and not the exact method of making it happen. Use cases once specified can be denoted both textual and visual representation. A key concept of use case modelling is that it helps us design a system from the end user's perspective. It is an effective technique for communicating system behaviour in the user's terms by specifying all externally visible system behaviour.

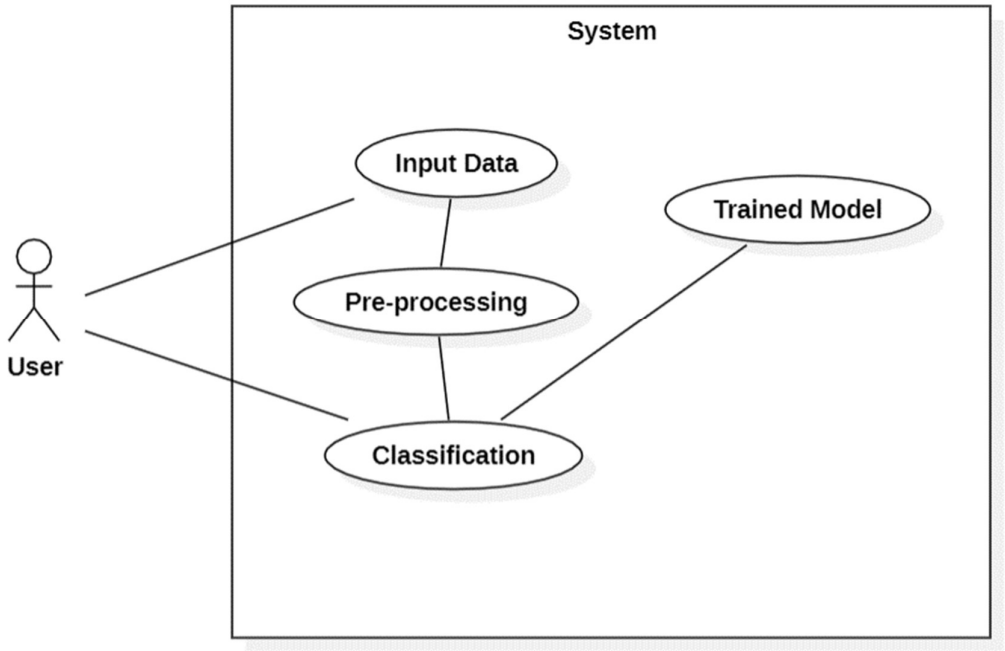

Figure 2 : Usecase Diagram

CLASS DIAGRAM:

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction. UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as –

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

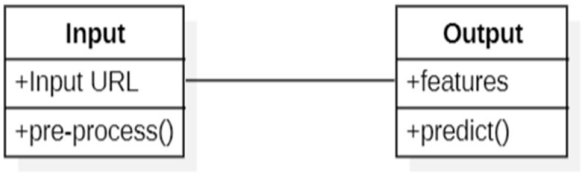

Figure 3 : Class Diagram

SEQUENCE DIAGRAM

To understand what a sequence diagram is, it's important to know the role of the Unified Modelling Language, better known as UML. UML is a modelling toolkit that guides the creation and notation of many types of diagrams, including behaviour diagrams, interaction diagrams, and structure diagrams.

A sequence diagram is a type of interaction diagram because it describes how—and in what order— a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process.

Sequence diagrams are sometimes known as event diagrams or event scenarios.

- To model high-level interaction among active objects within a system.
- To model interaction among objects inside a collaboration realizing a use case.
- It either model's generic interactions or some certain instances of interaction.

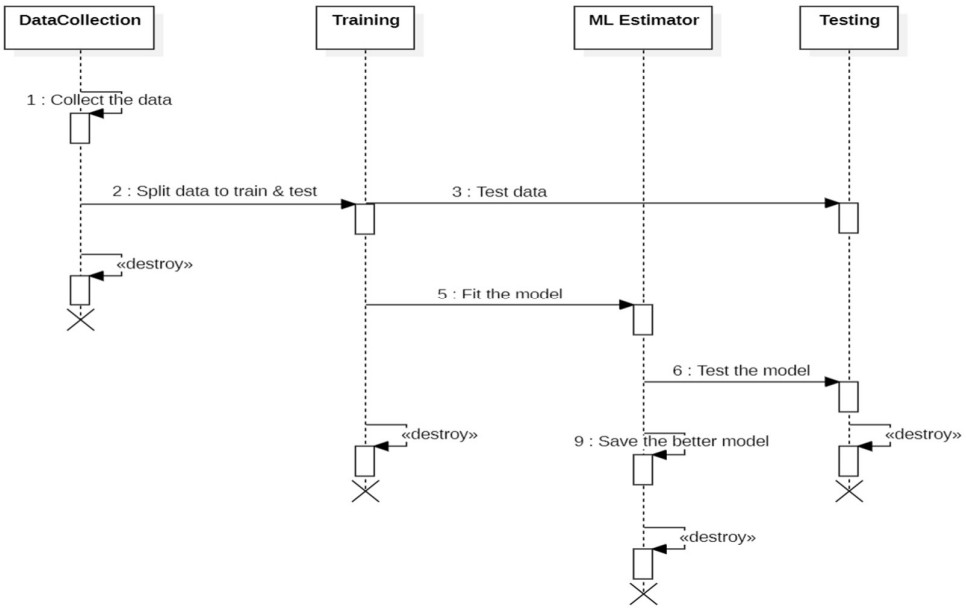

Figure 4 : Sequence Diagram

ACTIVITY DIAGRAM:

A UML activity diagram helps to visualize a certain use case at a more detailed level. It is a behavioural diagram that illustrates the flow of activities through a system. UML activity diagrams can also be used to depict a flow of events in a business process. They can be used to examine business processes in order to identify its flow and requirements.

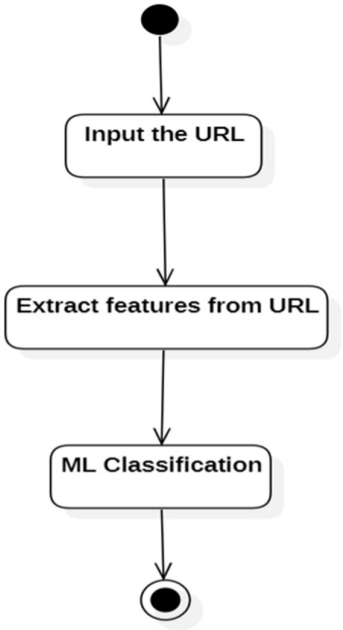

Figure 5 : Activity Diagram

CHAPTER – 5
TECHNOLOGIES

5.1. PYTHON

Python is a high level, interpreted and general-purpose dynamic programming language that focuses on code readability. It has fewer steps when compared to Java and C. It was founded in 1991 by developer Guido Van Rossum. It is used in many organizations as it supports multiple programming paradigms. It also performs automatic memory management.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Advantages:

- Presence of third-party modules
- Extensive support libraries (NumPy for numerical calculations, Pandas for data analytics, etc.)
- Open source and community development
- Easy to learn
- User-friendly data structures
- High-level language
- Dynamically typed language (No need to mention data type based on value assigned, it takes datatype)
- Object-oriented language
- Portable and Interactive
- Portable across Operating systems

Applications:

- GUI based desktop applications (Games, Scientific Applications)
- Web frameworks and applications
- Enterprise and Business applications
- Operating Systems
- Language Development
- Prototyping

Organizations using Python:

Some list of organisations that are using Python software for their software products are:

- Google (Components of Google spider and Search Engine)
- Yahoo (Maps)
- YouTube
- Mozilla
- Dropbox
- Microsoft
- Cisco
- Spotify
- Quora

5.2. REQUIRED PYTHON LIBRARIES

- Django
- Pandas
- Sklearn
- whois

Django:

Django is a free and open-source, Python-based web framework that follows the model–template–views (MTV) architectural pattern. It is maintained by the Django Software Foundation (DSF), an independent organization established in the US as a 501(c)(3) non-profit.

Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

Some well-known sites that use Django include Instagram, Mozilla, Disqus, Bitbucket, Nextdoor and Clubhouse.

Pandas:

Pandas is a software library written for the Python programming language for data manipulation and analysis. It offers data structures and operations for manipulating

numerical tables and time series. It is free software released under the three-clause BSD license. Pandas is a fast, powerful, flexible and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.

sklearn:

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a NumFOCUS fiscally sponsored project.

whois:

The whois library is a Python module facilitating access to WHOIS databases, allowing developers to programmatically retrieve domain registration details such as registrar information, registration and expiration dates, domain status, and owner contact information. This library is valuable for automating WHOIS queries, integrating domain information retrieval into Python applications, and managing domain-related data efficiently.

5.3. REACT JS

React.js, often just called React, is a free and widely used JavaScript library for building user interfaces. It allows you to create complex and dynamic UIs by breaking them down into smaller, reusable building blocks called components. Here's a closer look at how React works:

- **Components:** Imagine Lego bricks. React components are like these bricks - you can combine many small, independent components to create larger and more intricate UIs. Each component controls a specific part of the UI and can receive data through properties (props) to customize its appearance and behaviour.
- **State:** Components can also manage their own internal state, which is like private data specific to that component. If the state changes, the component automatically re-renders to reflect the update.
- **Virtual DOM:** For efficiency, React uses a virtual DOM, a lightweight in-memory representation of the actual UI in the browser. When a component's state or props change, React calculates the differences between the old and new

virtual DOM. This allows it to only update the parts of the real browser DOM that actually need changing, making things faster.

React offers several advantages:

- **Declarative:** You describe what the UI should look like, not how to manipulate it directly. This makes code cleaner and easier to maintain.
- **Component-Based:** Reusable components promote code organization and make complex UIs easier to manage.
- **Virtual DOM:** Enables efficient updates and improves performance.
- **Large Community:** React has a vast community and ecosystem of tools and libraries, providing extensive support for developers.

React has a learning curve, but its focus on components and clear separation of concerns makes it a powerful tool for building modern web applications. There are many resources available to learn React, including the official documentation and online courses.

CHAPTER – 6
IMPLEMENTATION

6.1. IMPLEMENTATION STEPS

To implement the idea of Phishing URL classification the following steps are mainly required:

- Data Collection & Dataset Description
- Feature Extraction
- Training Several Models
- Selecting the best model & model implementation

Let us explore every step-in detail,

6.1.1. DATA COLLECTION & DATASET DESCRIPTION

Dataset plays crucial role in training a machine to think like a human. The data of dataset should be accurate and trust worthy as much as possible. So, the data collection is carried out by visiting so many great dataset providers and finally I found a dataset from a great research dataset provider.

Aalto University, Finland is a great research provider which provides research data, research output, projects and many more., The dataset that is now being used is made available from 2014 by Sameul Marchal. The URI of this dataset is mentioned in 6 of References.

This dataset comes with 96,018 URLs with 12 features and class label. Among 96,018 URLs 48,009 are legitimate URLs where 48,009 are phishing URLs. The class label contains only two values, 0 and 1. Where 0 means legitimate and 1 means phishing URL.

6.1.2. FEATURE EXTRACTION

According to reference 1, features play a main role to detect a URL is phishing or not. So, the features have to be picked carefully. The features that I am going to use in this project are,

- **get_dot_count(url):** This function calculates and returns the number of dots (periods) present in the given URL, which can provide insights into the structure of the URL, such as domain names, subdomains, and file extensions.
- **get_url_length(url):** It determines the length of the URL string and returns the count of characters in the URL, indicating its overall length and potentially its complexity.

- **get_digit_count(url):** This function counts the number of digits present in the URL, offering information about whether the URL contains numerical identifiers or parameters.
- **get_special_char_count(url):** It counts and returns the number of special characters, such as semicolons, underscores, question marks, and ampersands, found within the URL, which can indicate non-standard or potentially suspicious elements.
- **get_hyphen_count(url):** Calculates and returns the count of hyphens ("-") present in the URL, which might suggest the usage of subdomains or complex URLs.
- **get_double_slash(url):** Determines the count of consecutive slashes ("//") in the URL, indicating whether the URL contains multiple slashes in sequence, which might be indicative of certain URL patterns or formats.
- **get_single_slash(url):** Counts the occurrence of single slashes ("/") in the URL, providing insights into the URL's structure and whether it follows standard path formats.
- **get_at_the_rate(url):** This function counts the occurrence of the "@" symbol in the URL, which is typically used for email addresses but may also appear in URLs for various purposes.
- **get_protocol(url):** Checks the URL's protocol (HTTP or HTTPS) and returns 1 if it's HTTP and 0 if it's HTTPS, helping identify the security protocol used in the URL.
- **get_protocol_count(url):** Counts the occurrences of "http" and "https" substrings in the URL, providing insights into the prevalence of these protocols and potential miscounts between HTTP and HTTPS instances.
- **get_registered_date_in_days(whois_result):** This function calculates the number of days between the current date and the registration date of the domain extracted from WHOIS data. Understanding the registration date is crucial in phishing URL classification because recent registrations may indicate suspicious or fraudulent activity, especially when combined with other features suggesting phishing behavior.
- **get_expiration_date_in_days(whois_result):** Determines the number of days remaining until the domain's expiration date based on WHOIS data. The

expiration date is essential in phishing URL classification as domains with short expiration periods or those already expired are often associated with malicious intent, indicating potential phishing attempts.

- **get_updated_date_in_days(whois_result):** Calculates the number of days since the last update to the domain registration information retrieved from WHOIS data. This feature is significant in phishing URL classification as frequent updates to registration details, particularly close to the current date, may suggest attempts to mask malicious activities or maintain anonymity, raising suspicion for potential phishing URLs.

6.1.3. TRAINING SEVERAL MODELS

Machine learning is such a huge field with several estimators included in it. Research is necessary to select the better model for every task of a machine learning. So, here I am going to use some classification algorithms to identify a better model. The classification algorithms I used are,

- Decision Tree
- KNN
- Kernal SVM
- Logistic Regression
- Naive Bayes
- Random Forest
- SVM

6.1.4. SELECTING THE BEST MODEL & MODEL IMPLEMENTATION

Based on the research and training results among all the above models, Random Forest performance seems good. The accuracies are as follows,

Decision Tree	0.874
KNN	0.8715
Kernel SVM	0.834
Log Regression	0.758
Naive Bayes	0.674
Random Forest	0.9035
SVM	0.781

Now Random Forest is going to be implemented over the project to get the expected classification results. The model is saved using pickle to load and use over project.

6.2. CODE MODULE

The feature extraction and API code in Django looks like,

```
from django.http import JsonResponse, HttpResponse
from django.views.decorators.csrf import csrf_exempt
import requests
from django.http import JsonResponse
from urllib.parse import urlparse
import datetime
import whois
from .settings import BASE_DIR
import os
import pickle

HTML_CONTENT = '''
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Phishing URL Detection</title>
</head>
<body>
<p>This URL is not intended for direct usage. It should only be accessed as an API.</p>
</body>
</html>
'''

def get_dot_count(url):
    return url.count('.')

def get_url_length(url):
    return len(url)

def get_digit_count(url):
```

```
    return sum(c.isdigit() for c in url)

def get_special_char_count(url):
    count = 0
    special_characters = [';', '+=', '_', '?', '=', '&', '[', ']']
    for each_letter in url:
        if each_letter in special_characters:
            count = count + 1
    return count

def get_hyphen_count(url):
    return url.count('-')

def get_double_slash(url):
    return url.count('//')

def get_single_slash(url):
    return url.count('/')

def get_at_the_rate(url):
    return url.count('@')

def get_protocol(url):
    protocol = urlparse(url)
    return 1 if protocol.scheme == 'http' else 0

def get_protocol_count(url):
    http_count = url.count('http')
    https_count = url.count('https')
    http_count = http_count - https_count
    return (http_count + https_count)

def perform_whois(url):
    try:
```

```
        whois_result = whois.whois(url)
        return whois_result
    except Exception as e:
        return False

def get_registered_date_in_days(whois_result):
    if (whois_result != False):
        created_date = whois_result.creation_date
        if ((created_date is not None) and (type(created_date) != str)):
            if (type(created_date) == list):
                created_date = created_date[0]
            today_date = datetime.datetime.now()
            days = (today_date-created_date).days
            return days
        else:
            return -1
    else:
        return -1

def get_expiration_date_in_days(whois_result):
    if (whois_result != False):
        expiration_date = whois_result.expiration_date
        if ((expiration_date is not None) and (type(expiration_date) != str)):
            if (type(expiration_date) == list):
                expiration_date = expiration_date[0]
            today_date = datetime.datetime.now()
            days = (expiration_date-today_date).days
            return days
        else:
            return -1
    else:
        return -1

def get_updated_date_in_days(whois_result):
```

```
    if (whois_result != False):
        updated_date = whois_result.updated_date
        if ((updated_date is not None) and (type(updated_date) != str)):
            if (type(updated_date) == list):
                updated_date = updated_date[0]
            today_date = datetime.datetime.now()
            days = (today_date-updated_date).days
            return days
        else:
            return -1
    else:
        return -1

def get_result(features):
    model_path = os.path.join(BASE_DIR, "Models/random_forest_calssifier.pkl")
    model = pickle.load(open(model_path, "rb"))
    return model.predict([features])

def extract_features(url):
    whois_result = perform_whois(url)
    features = [get_registered_date_in_days(whois_result),
                get_expiration_date_in_days(whois_result),
                get_updated_date_in_days(whois_result),
                get_dot_count(url),
                get_url_length(url),
                get_digit_count(url),
                get_special_char_count(url),
                get_hyphen_count(url),
                get_double_slash(url),
                get_single_slash(url),
                get_at_the_rate(url),
                get_protocol_count(url)
                ]
    return features
```

```
def get_data(url):
    try:
        response = requests.get(url, allow_redirects=True)
    except Exception as e:
        return {"exception": str(e)}
    features = extract_features(url)
    prediction = get_result(features)[0]
    json_data = {
        "Requested_URL": url,
        "Destinaton_URL": response.url,
        "Registered_Date_in_Days": features[0],
        "Expiration_Date_in_Days": features[1],
        "Updated_Date_in_Days": features[2],
        "Dot_Count": features[3],
        "URL_Length": features[4],
        "Digit_Count": features[5],
        "Special_Char_Count": features[6],
        "Hyphen_Count": features[7],
        "Double_Slash_Count": features[8],
        "Single_Slash_Count": features[9],
        "At_The_Rate_Count": features[10],
        "Protocol_Count": features[11],
        "prediction": "Safe" if prediction == 1 else "Dangerous"
    }
    return json_data

@csrf_exempt
def index(request):
    if request.method == "POST":
        url = request.POST["URL"]
        return JsonResponse(get_data(url))
    return HttpResponse(HTML_CONTENT)
```

CHAPTER – 7
TESTING

7.1. TESTING

System testing involves user training system testing and successful running of the developed proposed system. The user tests the developed system and changes are made according to their needs. The testing phase involves the testing of developed system using various kinds of data.

An elaborate testing of data is prepared and the system is tested using the test data. While testing, errors are noted and the corrections are made. The corrections are also noted for the future use. The users are trained to operate the developed system.

Testing involves operation of a system or application under controlled conditions and evaluating the results. The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should .it is oriented to 'detection'.

System testing is the stage of implementation that is aimed at ensuring that the system works accurately before live operation commences. Testing is vital to the success of the system. System testing makes logical assumption that if all the parts of the system are correct, then the goal will be successfully achieved. A series of testing are done for the proposed system before the system is ready for the user.

7.2. TESTING METHODOLOGIES

There are two major types of testing. They are

1. White Box Testing
2. Black Box Testing

7.2.1. WHITE BOX TESTING

White box testing (also known as clear box testing, glass box testing, and transparent box testing and structural testing) is a method of testing software that tests internal structures are working of an application, an opposed to its functionality. In white box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tests are choosing inputs to exercise paths through the code and determine outputs. While white box testing can be applied at the unit, integration and system level of the software testing process, it is usually done at the unit level. It can test paths within a unit, path between units during integration, and between sub systems during system level tests.

This method of test design can uncover many of our problems it might not detect unimplemented parts of the specification or missing requirements.

White box test design techniques include:

- Control flow testing
- Data flow testing
- Branch testing
- Path testing
- Statement coverage
- Decision coverage

7.2.2 BLACK BOX TESTING

Black box testing is a method of software testing that examines the functionality of an (see white box testing). This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically compares most if not all higher-level testing but can also dominate unit testing as well.

The following are the types of testing:

1. Unit Testing
2. Integration Testing
3. Validation Testing
4. Verification Testing
5. Acceptance Testing

1. Unit Testing

Unit testing focuses verification efforts and the smallest unit of the software design, the module. This is also known as “module testing”. The modules are tested separately, this testing was carried out during programming stage itself, in this testing each module is found to be working satisfactory as regards to the expected output from the module.

2. Integration Testing

Data can be lost across an interface: one module can have adverse efforts on another. Integration testing is the systematic testing for construction of program structure, while at the same time conducting test to uncover errors associated with in the interface. A correction is difficult because the isolation of the cause is complicated by the cast expanse of the entire program. Thus, in the integration testing step, all the errors uncovered are corrected for the next testing step.

3. Validation Testing

At the conclusion of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and corrected and a final series of software tests begins validation test has been conducted of the two possible conditions exists. One is the function or performance characteristics confirmed to specifications and are accepted and the other is deviation from specifications in uncovered and efficiency list is created.

4. Verification Testing

Verification is a fundamental concept in software design. This is the bridge between customer requirements and an implementation that specifies those requirements. This is verifiable if it can be demonstrated that the testing will result in an implementation that satisfies the customer requirements.

5. User Acceptance Testing

User acceptance testing of a system is the key factor of the success of the nay system. The system under study is tested for the user acceptance by constantly keeping in touch with their susceptive system users at any time of developing and making changes whenever required.

CHAPTER – 8
SCREENSHOTS

Homepage:

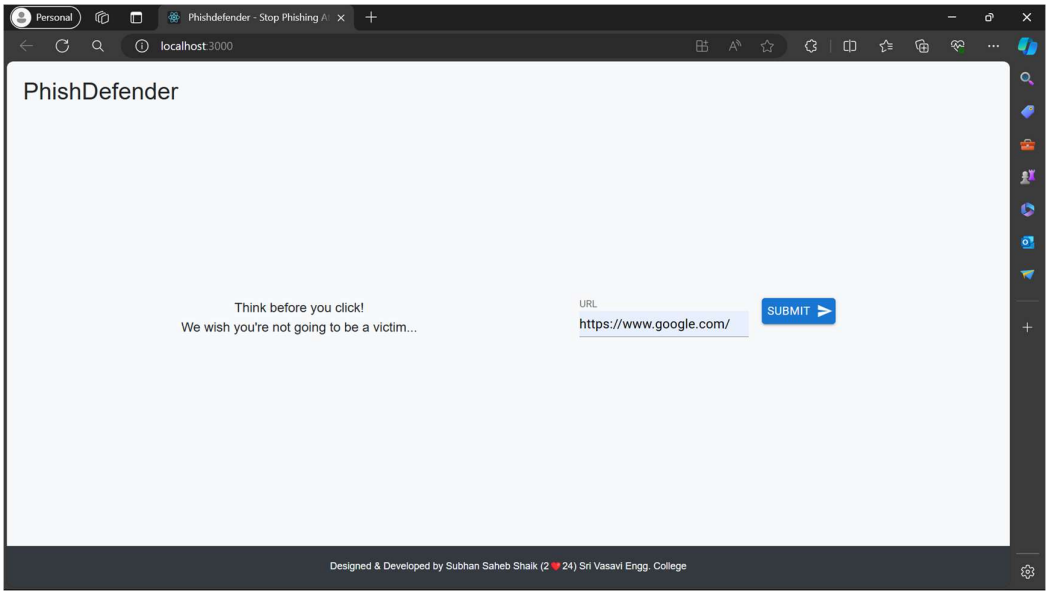

Figure 6 : Home Page

Result Page:

Legitimate/Safe URL:

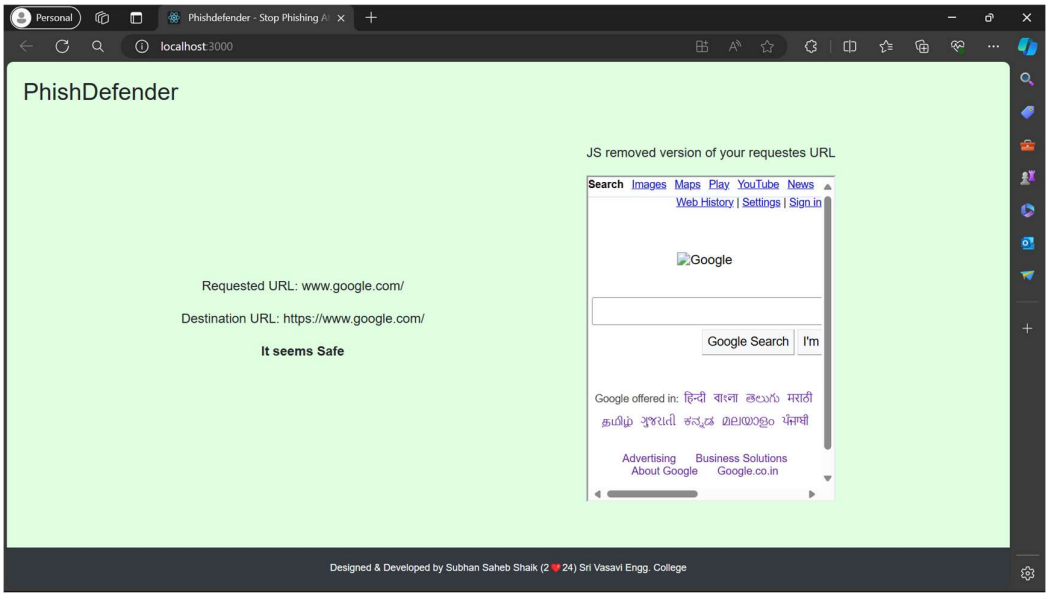

Figure 7 : Result Page - Legitimate URL Case

Phishing URL/Error Case:

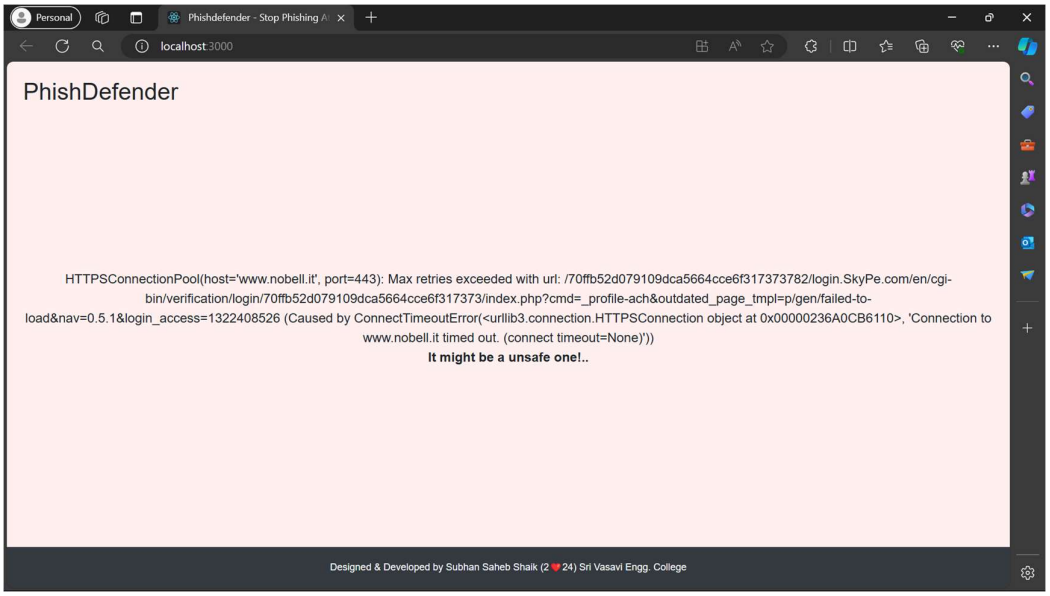

Figure 8 : Result Page - Unsafe URL Case

Special Access from anywhere (API):

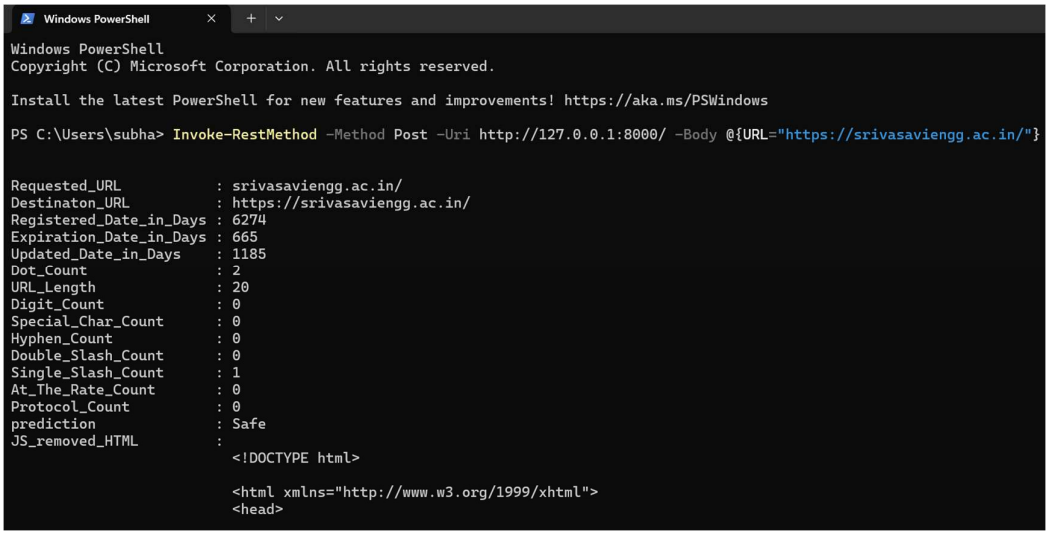

Figure 9 : API Accessing through Terminal (Windows)

CHAPTER – 9
CONCLUSION

CONCLUSION

After completing this comprehensive project and conducting research, I had successfully achieved all my objectives and developed a user-friendly interface. Utilizing machine learning techniques, I have created a robust system capable of classifying URLs based on various features such as URL length, dot count, and other relevant parameters.

One of the primary goals of this project was to provide access to machine learning-based URL classification through an API endpoint. By implementing this feature, users can access our application's services from a terminal, a Python script, or any other programming environment capable of interacting with APIs.

In conclusion, this project represents a significant achievement in the field of cybersecurity, combining machine learning, web development, and API integration to deliver a valuable and practical solution for URL classification. We are confident that our application will serve as a useful tool for users seeking to enhance their online security measures.

FUTURE SCOPE

Future developments could focus on real-time monitoring and dynamic updating of classification models based on evolving threats and patterns. This would involve integrating machine learning models with continuous data streams to adapt to new threats and ensure proactive threat detection.

Furthermore, exploring techniques such as deep learning and reinforcement learning could lead to more sophisticated and adaptive URL classification systems. These advanced AI algorithms have the potential to learn complex patterns and relationships in data, further enhancing the accuracy and effectiveness of URL classification.

In summary, the future of this project lies in leveraging advanced AI algorithms, particularly NLP, deep learning, and reinforcement learning, to enhance URL classification capabilities. By incorporating these techniques, we can develop more robust and intelligent systems for detecting and mitigating online threats, ultimately improving cybersecurity for users worldwide.

CHAPTER – 10

REFERENCES

[1] M. Zabihimayvan and D. Doran, “Fuzzy Rough Set Feature Selection to Enhance Phishing Attack Detection”, 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), New Orleans, LA, USA, 2019, pp. 1-6, DOI: 10.1109/FUZZ-IEEE.2019.8858884.

[2] I. Kara, M. Ok and A. Ozaday, “Characteristics of Understanding URLs and Domain Names Features: The Detection of Phishing Websites with Machine Learning Methods”, in IEEE Access, vol. 10, pp. 124420-124428, 2022, DOI: 10.1109/ACCESS.2022.3223111.

[3] S. Parekh, D. Parikh, S. Kotak and S. Sankhe, “A New Method for Detection of Phishing Websites: URL Detection”, 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, 2018, pp. 949-952, DOI: 10.1109/ICICCT.2018.8473085.

[4] M. M. Vilas, K. P. Ghansham, S. P. Jaypralash and P. Shila, “Detection of Phishing Website Using Machine Learning Approach”, 2019 4th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT), Mysuru, India, 2019, pp. 384-389, DOI: 10.1109/ICEECCOT46775.2019.9114695.

[5] N. F. Abedin, R. Bawm, T. Sarwar, M. Saifuddin, M. A. Rahman and S. Hossain, “Phishing Attack Detection using Machine Learning Classification Techniques”, 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), Thoothukudi, India, 2020, pp. 1125-1130, doi: 10.1109/ICISS49785.2020.9315895.

[6] https://research.aalto.fi/en/datasets/phishstorm-phishing-legitimate-url-dataset

[7] Basnet, R., Mukkamala, S., Sung, A.H. (2008). Detection of Phishing Attacks: A Machine Learning Approach. In: Prasad, B. (eds) Soft Computing Applications in Industry. Studies in Fuzziness and Soft Computing, vol 226. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-77465-5_19

[8] J. Kumar, A. Santhanavijayan, B. Janet, B. Rajendran and B. S. Bindhumadhava, "Phishing Website Classification and Detection Using Machine Learning," 2020 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2020, pp. 1-6, doi: 10.1109/ICCCI48352.2020.9104161.

[9] A. Alswailem, B. Alabdullah, N. Alrumayh and A. Alsedrani, "Detecting Phishing Websites Using Machine Learning," 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 2019, pp. 1-6, doi: 10.1109/CAIS.2019.8769571.