VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi-590018

A

Mini Project Report

On

"Detection of Phishing Domains Using AIML"

SUBMITTED IN PARTIAL FULFILLMENT FOR 6TH SEMESTER

BACHELOR OF ENGINEERING

IN

COMPUTER SCEINCE AND ENGINEERING

SUBMITTED BY

Indhu Shriya K R (1JB21CS060)
Mohan Gowda T (1JB21CS090)
Naga Balaji K N (1JB21CS093)
Nithu Shree (1JB21CS101)

Under the Guidance of

Dr. Roopa M J
Associate Professor
Dept of CSE

SJBIT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SJB INSTITUTE OF TECHNOLOGY

No.67, BGS Health & Education City, Dr.Vishnuvardhan Rd, Kengeri, Bengaluru, Karnataka 560060

An Autonomous Institute under VTU

Approved by AICTE - New Delhi, Accredited by NAAC A+, Accredited by NBA

2023 - 2024

|| Jai Sri Gurudev ||
Sri Adichunchanagiri Shikshana Trust ®

SJB INSTITUTE OF TECHNOLOGY

No.67, BGS Health & Education City, Dr.Vishnuvardhan Rd, Kengeri, Bengaluru, Karnataka 560060
An Autonomous Institute under VTU
Approved by AICTE - New Delhi, Accredited by NAAC A+, Accredited by NBA

Department of Computer Science and Engineering

SJBIT

CERTIFICATE

Certified that the Mini Project entitled "**Detecting Phishing Domains Using AIML**" carried out by **Indhu Shriya K R, Mohan Gowda T, Naga Balaji K N, Nithu Shree** bearing USN **1JB21CS060, 1JB21CS090, 1JB21CS093, 1JB21CS101** are bonafide students of **SJB Institute of Technology** in partial fulfilment for 6th semester of **BACHELOR OF ENGINEERING** as prescribed by **Visvesvaraya Technological University**, **Belagavi** during the academic year **2023-24.** It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the Departmental library. The project report has been approved as it satisfies the academic requirements in respect of Mini Project prescribed for the said Degree.

Signature of Guide	**Signature of HOD**
Dr. Roopa M J	**Dr. Krishna A N**
Associate Professor	**Professor & Head**
Dept. of CSE, SJBIT	**Dept. of CSE, SJBIT**



ACKNOWLEDGEMENT

We would like to express our profound grateful to His Divine Soul **Jagadguru Padmabhushan Sri Sri Sri Dr. Balagangadharanatha Mahaswamiji** and His Holiness **Jagadguru Sri Sri Sri Dr. Nirmalanandanatha Mahaswamiji** for providing us an opportunity to complete our academics in this esteemed institution.

We would also like to express our profound thanks to **Revered Sri Sri Dr. Prakashnath Swamiji**, Managing Director, BGS & SJB Group of Institutions & Hospitals, for his continuous support in providing amenities to carry out this Mini Project in this admired institution.

We express our gratitude to **Dr. K. V. Mahendra Prashanth,** Principal, SJB Institute of Technology, for providing us an excellent facilities and academic ambience; which have helped us in satisfactory completion of Mini Project.

We extend our sincere thanks to **Dr. Krishan A N,** Head of the Department, Computer Science and Engineering for providing us an invaluable support throughout the period of our Mini Project.

We wish to express our heartfelt gratitude to our Guide and Mini Project Coordinator, **Dr. Roopa M J,** for her valuable guidance, suggestions and cheerful encouragement during the entire period of our Mini Project.

Finally, we take this opportunity to extend our earnest gratitude and respect to our parents, Teaching & Non teaching staffs of the department, the library staff and all our friends, who have directly or indirectly supported us during the period of our Mini Project work.

Regards,
Indhu Shriya K R [1JB21CS060]
Mohan Gowda T [1JB21CS090]
Naga Balaji K N [1JB21CS093]
Nithu Shree [1JB21CS101]

ABSTRACT

Phishing attacks represent a significant and growing threat to cybersecurity, as they aim to deceive users into divulging sensitive information through fraudulent domains. This project seeks to develop a robust detection system for phishing domains utilizing artificial intelligence and machine learning techniques. The approach begins with the collection of an extensive dataset comprising both phishing and legitimate URLs, sourced from various online repositories and APIs. Preprocessing steps ensure the data is clean and transform raw URLs into feature vectors suitable for machine learning models.

Feature extraction is a critical component of the detection process. Key features such as URL length, presence of special characters, use of HTTPS, domain age, and other relevant attributes are systematically identified and extracted using an automated feature extraction module. Multiple machine learning algorithms, including logistic regression, decision trees, random forests, and neural networks, are explored to identify the most effective model for phishing detection. These algorithms are rigorously evaluated based on performance metrics such as accuracy, precision, recall, and F1-score.

The most accurate model is selected, fine-tuned, and validated using cross-validation techniques to ensure its reliability and robustness. The implementation of the detection system is carried out using Python and TensorFlow, with an API module integrating the trained model to facilitate real-time predictions. This module accepts a URL as input, extracts its features, and outputs the probability of it being a phishing domain.

Experimental results indicate that the model achieves high precision and recall in distinguishing phishing domains from legitimate ones, demonstrating its effectiveness in enhancing cybersecurity measures. The developed system is scalable and can be integrated into existing security frameworks, offering a valuable tool for real-time phishing detection. This project underscores the potential of AI and ML in combating phishing threats and contributing to safer online experiences for users.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In today's interconnected digital world, the rise of internet usage has been accompanied by an increase in cyber threats, particularly phishing attacks. Phishing is a deceptive attempt to obtain sensitive information such as usernames, passwords, and credit card details by masquerading as a trustworthy entity in electronic communications. As these attacks grow in sophistication, there is an escalating need for effective solutions to detect and prevent them, ensuring the safety and security of online users.

Phishing attacks are a significant concern for individuals and organizations alike. They often come in the form of deceptive emails, messages, or websites that lure users into divulging personal information. The consequences of falling for these scams can be severe, leading to financial loss, identity theft, and unauthorized access to sensitive data. Despite the widespread awareness of phishing, the tactics used by attackers continue to evolve, making it challenging for users to recognize and avoid them.

The need to address phishing attacks is pressing. Phishing not only targets individuals but also poses a considerable threat to businesses and governmental organizations. The financial sector, in particular, faces a high risk due to the sensitive nature of the information it handles. As digital transactions become more prevalent, the potential impact of phishing attacks increases, highlighting the importance of developing robust countermeasures.

Cybersecurity has become a critical aspect of modern life, with the rapid advancement of technology creating new opportunities for both legitimate and malicious activities. In this context, the detection and prevention of phishing attacks have emerged as vital components of a comprehensive cybersecurity strategy. The complexity of these attacks necessitates sophisticated and adaptive approaches that can keep up pace with the evolving threat landscape.

1.1 About the project

The primary goal of this project is to develop a system capable of detecting malicious links and protecting users from phishing attacks. This involves creating a framework that can accurately identify and block harmful URLs before they cause any damage. By leveraging advanced technologies and innovative methodologies, this project seeks to provide a reliable and effective solution to a pervasive problem.

In addressing phishing attacks, it is essential to understand the various forms they can take. Phishing techniques range from simple deceptive emails to highly targeted and

personalized attacks known as spear-phishing. These tactics exploit human psychology, relying on fear, urgency, and curiosity to manipulate victims into revealing sensitive information. The development of countermeasures must consider the diverse nature of these attacks to offer comprehensive protection.

One of the key challenges in detecting phishing attacks is the rapid evolution of phishing tactics. Attackers continually adapt their methods to circumvent traditional security measures, making it difficult to rely on static rules and signatures for detection. This dynamic nature of phishing necessitates the use of adaptive and intelligent systems that can learn from new data and detect previously unseen threats.

The project aims to develop a system that can analyze URLs and identify potential phishing threats. This involves examining various characteristics of URLs, such as their structure, content, and hosting details, to determine their legitimacy. By leveraging machine learning and heuristic analysis, the system can learn to recognize patterns and anomalies associated with phishing attacks, improving its detection capabilities over time.

Another critical aspect of the project is the need for user-friendly solutions. Effective phishing detection tools must be accessible and easy to use for non-technical users. This ensures that the benefits of advanced cybersecurity measures are available to a broad audience, enhancing overall protection. The development of intuitive interfaces and clear warning mechanisms is essential to achieving this goal.

The significance of this project extends beyond individual users. By providing a reliable tool for detecting phishing attempts, the project contributes to the broader effort of safeguarding sensitive information and reducing the risk of cyber fraud. This proactive approach not only helps in mitigating immediate threats but also in building a more secure digital environment for the future. Organizations can benefit from enhanced security measures, reducing the risk of data breaches and maintaining the trust of their customers.

In addition to protecting against phishing attacks, the project has the potential to contribute to the advancement of cybersecurity research. By developing and refining new techniques for detecting malicious links, the project can provide valuable insights and methodologies that can be applied to other areas of cybersecurity. This includes the detection of other types of cyber threats, such as malware and ransomware, further enhancing the overall security landscape.

The development of a robust phishing detection system involves several stages. Initially, it requires the collection of a diverse dataset of URLs, including both legitimate and malicious examples. This dataset forms the basis for training and evaluating the detection

algorithms. The quality and diversity of the dataset are crucial for developing a system that can generalize well to real-world scenarios.

Feature extraction is a critical step in the development process. This involves identifying and extracting relevant features from the URLs that can help distinguish between legitimate and malicious links. These features may include lexical characteristics, such as the length of the URL and the presence of suspicious keywords, as well as host-based features, such as domain registration details and IP address information. Content-based features, such as the presence of certain HTML elements, can also be important indicators of phishing.

Machine learning algorithms play a central role in the detection system. These algorithms are trained on the extracted features to learn patterns associated with phishing attacks. Various machine learning techniques, such as decision trees, random forests, and neural networks, can be employed to build and refine the detection model. The choice of algorithm and its configuration are crucial for achieving high accuracy and low false positive rates.

Evaluation is a critical part of the development process. The performance of the detection system must be thoroughly tested using a separate validation dataset. Key metrics, such as accuracy, precision, recall, and F1 score, are used to assess the effectiveness of the system. These metrics provide a comprehensive view of the system's ability to detect phishing URLs while minimizing false alarms.

Once the detection system is validated, it can be integrated into a user-friendly tool or application. This tool provides real-time protection by analyzing URLs and warning users about potential phishing threats. The development of an intuitive interface and clear warning mechanisms is essential for ensuring that users can effectively utilize the tool without requiring advanced technical knowledge.

The implementation of a phishing detection system also involves considerations of scalability and performance. The system must be capable of handling large volumes of URLs in real-time, ensuring that users receive timely warnings. This requires efficient algorithms and optimized code to maintain high performance while providing accurate detection.

The development of an effective system to detect malicious links and avoid phishing attacks is a critical step in the ongoing battle against cybercrime. By leveraging advanced technologies and innovative approaches, this project aims to provide a robust and reliable solution to protect users from the ever-evolving threat of phishing. The project's significance lies in its potential to enhance cybersecurity measures, safeguard sensitive information, and contribute to the broader effort of creating a more secure digital environment.

The comprehensive approach taken by this project addresses the multifaceted nature of phishing attacks, combining technical innovation with user-centric design. By developing a system that is both effective and accessible, the project ensures that the benefits of advanced cybersecurity measures are available to a wide audience. This not only helps in mitigating immediate threats but also in building a foundation for future advancements in cybersecurity.

As the digital landscape continues to evolve, the importance of proactive and adaptive security measures cannot be overstated. The development of systems capable of detecting and preventing phishing attacks is a vital component of a comprehensive cybersecurity strategy. This project represents a significant contribution to this effort, providing a reliable and effective solution to one of the most pervasive threats in the digital world.

In the long term, the insights and methodologies developed through this project can inform the broader field of cybersecurity research. By exploring new techniques for detecting malicious links, the project can provide valuable knowledge that can be applied to other areas of cybersecurity. This includes the detection of other types of cyber threats, such as malware and ransomware, further enhancing the overall security landscape.

This project is to create a safer and more secure digital environment for all users. By addressing the pressing issue of phishing attacks, the project aims to protect individuals and organizations from the significant risks associated with cyber threats. Through the development of advanced detection systems and user-friendly tools, this project contributes to the ongoing effort to build a more resilient and secure digital world.

CHAPTER 2

LITERATURE SURVEY

AUTHOR	YEAR	TITLE	METHODOLOGY	DRAWBACK
Abbas Jabr Saleh Albahadili, Ayhan Akbas	2024	Detection of phishing URLs with deep learning based on GAN-CNN-LSTM network and swarm intelligence algorithms	Balancing datasets with GAN-generated synthetic samples, extracting and selecting features using CNN and the White Shark Optimization algorithm, and classifying phishing attacks with an LSTM neural network.	otential computational complexity and scalability issues of the CNN-WSO-LSTM method, which may limit its real-time application in phishing detection.
Lizhen tang and qusay h. Mahmoud	2021	A Deep Learning-Based Framework for Phishing Website Detection	Describes a classification approach to phishing website detection using AI meta-learners and base learners.Details on the dataset, feature extraction, and model training processes are provided.	Limitations of single classifier methods.High false alarm rate and low detection rate in some models
Mehmet Korkmaz,Ozgur Koray Sahingoz, Banu Diri	2020	Detection of Phishing Websites by Using Machine Learning-Based URL Analysis	Focuses on the implementation of AI meta-learners (AdaBoost, Bagging, Rotation Forest, LogitBoost) combined with the Extra-trees algorithm.The dataset preparation, model training, and evaluation processes are outlined.	Challenges in handling dynamic and evolving phishing attacks. The need for high accuracy and low false positive rates in detection models.
Y. A. Alsariera, V. E. Adeyemo, A. O. Balogun, A. K. Alazzawi	2020	AI Meta-Learners and Extra-Trees Algorithm for the Detection of Phishing Websites	Utilizes AI meta-learners (Bagging, AdaBoost, Rotation Forest, LogitBoost) and the Extra-trees algorithm.10-fold cross-validation with metrics: ROC, accuracy, false positive rate, and F-measure.	High false positive and negative rates.The inability of single classifiers to effectively detect evolving phishing websites
Junaid Rashid, Tahira Nazir, Toqeer	2020	Phishing Detection Using Machine Learning Technique	Collected web pages using GNU Wget and Python scripts.Extracted feature vectors from URLs using vocabulary, host, word-	Need for feature extraction and high data dimensionality.Dependence on

Mahmood, Muhammad Wasif Nisar			based features, and third-party services.Applied Principal Component Analysis (PCA) to reduce dimensionality.	dataset quality and variety. Potential inaccuracy in real-time applications due to evolving phishing tactics.

Several research papers have focused on phishing detection using various machine learning and deep learning approaches. The 2024 study by Albahadili and Akbas introduces a CNN-WSO-LSTM framework that balances datasets with GAN-generated samples, selects features with the White Shark Optimization algorithm, and classifies phishing attacks using LSTM. However, it may face challenges in real-time applications due to potential computational complexity and scalability issues [1].

In 2021, Tang and Mahmoud developed a deep learning-based framework using AI meta-learners for phishing website detection. While their approach provides a detailed classification methodology, it suffers from high false alarm rates and low detection accuracy in some models, particularly in single-classifier methods [2].

The 2020 research by Korkmaz, Sahingoz, and Diri focused on using AI meta-learners combined with the Extra-trees algorithm to detect phishing websites. Despite its comprehensive dataset preparation and model training processes, this method struggles with dynamic and evolving phishing attacks, requiring higher accuracy and lower false positive rates [3].

Another 2020 study by Alsariera and colleagues explored AI meta-learners like Bagging and AdaBoost with the Extra-trees algorithm for phishing detection. The study highlighted the limitations of single classifiers, which often result in high false positive and negative rates, making them less effective against evolving phishing tactics [6].

Lastly, Rashid and his team, in their 2020 paper, used machine learning techniques to detect phishing by extracting features from URLs and applying PCA for dimensionality reduction. However, the reliance on dataset quality and the evolving nature of phishing attacks pose challenges, potentially leading to inaccuracies in real-time applications [9]. Overall, these studies highlight the ongoing efforts and challenges in improving phishing detection accuracy and reducing false positives.

CHAPTER 3

PROBLEM STATEMENT

Phishing attacks pose a significant and growing threat in the digital world, deceiving individuals and organizations into divulging sensitive information. Existing detection methods often fall short due to the rapidly evolving tactics used by cybercriminals. A more sophisticated and adaptive solution is urgently needed to effectively identify and mitigate these threats.Phishing attacks exploit human psychology and leverage increasingly sophisticated techniques to trick users into revealing confidential information.

3.1 Challenges

Traditional security measures have several drawbacks that limit their effectiveness:

1. Static Rule-Based Systems: These systems rely on predefined rules and signatures to detect phishing attempts. Cybercriminals continually adapt their tactics to bypass these static defenses, leading to high false positive and false negative rates.

2. Limited Feature Analysis: Many existing models focus on a narrow set of features, such as URL structure or content. This limited scope can result in missed threats, as phishing techniques can vary widely.

3. Performance Issues: Some models struggle with processing large volumes of data in real-time, leading to delays in threat detection and response.

4.Insufficient Real-Time Updates: Many existing systems are slow to update their databases with new threats, which means that users can be exposed to phishing attacks that have not yet been recognized by the system.

3.2 Objectives of the project

Our project addresses these challenges by developing a unique system that utilizes a Feed Forward Neural Network (FFNN) to detect malicious links and prevent phishing attacks. This approach offers several advantages over existing models. The objectives we have set for the project are:

1. To design a model that continuously learns from new phishing data and improves its accuracy in detecting phishing domains over time.
2. To analyze the comprehensive features extracted to examine a wide range of URL characteristics, including lexical, host-based, and content-based features.

3. To develop a FFNN model that is efficient and scalable, capable of processing larger datasets and detecting a wide range of malicious links.

4. To recognize that malicious links take new forms every day, enabling the model to understand patterns in phishing links.

CHAPTER 4

REQUIEMENT ANALYSIS

4.1 Software Requirements

1. Operating System:

- Windows 10 or later
- macOS Catalina or later
- Linux (Ubuntu 18.04 or later)

2. Programming Languages:

- Python 3.7 or later

3. Libraries and Frameworks:

- TensorFlow or PyTorch (for implementing the Feed Forward Neural Network)
- Scikit-learn (for machine learning algorithms and data processing)
- Pandas (for data manipulation and analysis)
- NumPy (for numerical computations)
- Matplotlib or Seaborn (for data visualization)
- Flask or Django (for developing the web interface)
- Requests (for making HTTP requests and handling URLs)

4. Development Tools:

- Jupyter Notebook or JupyterLab (for interactive development and experimentation)
- Integrated Development Environment (IDE) such as PyCharm, Visual Studio Code

5. Database Management System:

- MySQL, PostgreSQL, or SQLite (for storing URL data and other related information)

6. Version Control System:

- Git (for source code management and collaboration)

7. Web Browser:

- Google Chrome, Mozilla Firefox, or Microsoft Edge (for testing the web interface)

4.2 Hardware Requirements

1. Processor:

- Intel Core i5 or AMD Ryzen 5 equivalent or higher

2. Memory (RAM):

- Minimum 8 GB (16 GB or more recommended for large datasets and faster processing)

3. Storage:

- Minimum 256 GB SSD (Solid State Drive) for faster read/write operations
- Additional HDD or SSD storage for large datasets, if needed

4.Graphics Processing Unit (GPU) (for training neural networks):

- NVIDIA GeForce GTX 1050 or higher with CUDA support
- Compatible with TensorFlow or PyTorch GPU support

5. Network:

- Stable internet connection for downloading libraries, datasets, and updates
- High-speed internet for accessing and analyzing URLs in real-time

6. Peripherals:

- Monitor with a resolution of at least 1920x1080
- Keyboard and mouse for development work
- External hard drive for backup and additional storage

CHAPTER 5

DESIGN AND ARCHITECTURE

This chapter provides a comprehensive overview of the design and architecture of a phishing detection system using a Feedforward Neural Network (FNN). It outlines the entire workflow, starting from the identification and retrieval of datasets, which are essential for training the model. The chapter covers the data preparation process, including data processing, feature extraction, and scaling, ensuring the data is suitable for modeling. The FNN is highlighted as the core machine learning algorithm, tasked with learning patterns to accurately classify URLs as phishing or legitimate. The chapter also discusses the evaluation and tuning of the model to optimize performance, followed by its deployment and continuous monitoring in a real-world environment. The iterative nature of the process is emphasized, allowing for continuous improvement and adaptation to evolving phishing tactics.

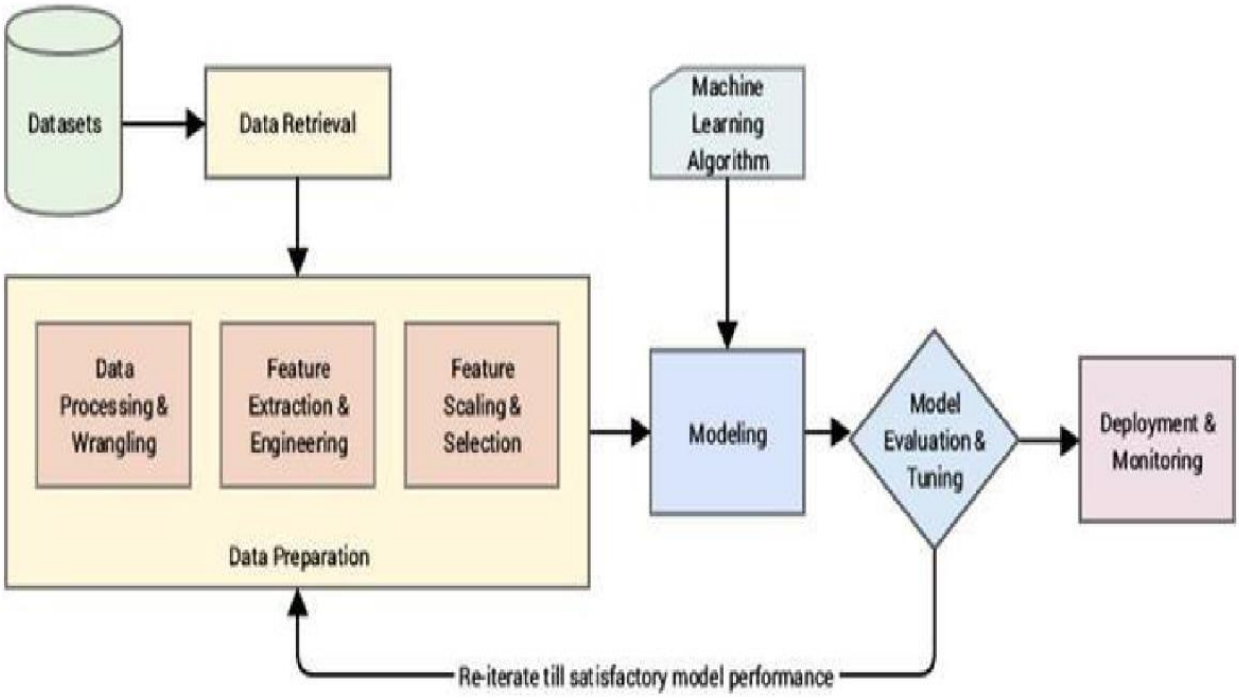

Fig 5.1 Workflow diagram

Fig 5.1 shows the workflow diagram outlines the process of phishing detection using a Feedforward Neural Network (FNN). It starts with data retrieval from datasets, followed by data preparation, which includes processing, feature extraction, and feature selection. The prepared data is then fed into a machine learning algorithm for modeling, with iterative

evaluation and tuning until satisfactory performance is achieved, after which the model is deployed and monitored.

1.Datasets

The process begins with identifying and utilizing appropriate datasets. These datasets are the foundation of the project, containing the raw data needed to train the detection model. The data typically includes URLs, along with features and labels that indicate whether they are malicious or legitimate.

2.Data-Retrieval

Once the datasets are selected, the next step is data retrieval. This involves collecting the necessary data from the datasets, which may require querying databases or reading data from files. This step ensures that all relevant data is gathered for further processing.

2.1 Data Processing & Wrangling

After retrieving the data, it undergoes initial processing and wrangling. This stage involves cleaning the data, addressing any missing values, and formatting it to make it suitable for analysis. Proper data preparation is crucial, as it ensures that the subsequent steps of feature extraction and modeling are built on a solid foundation.

2.2 Feature Extraction & Engineering

In this phase, features are extracted and engineered from the raw data. The goal is to identify and derive relevant features that the machine learning model will use. These features might include lexical characteristics of URLs, host-based attributes, and content-based indicators. Feature engineering is vital for capturing the nuances that distinguish phishing URLs from legitimate ones.

2.3 Feature Scaling & Selection

Following feature extraction, the data undergoes scaling and selection. Feature scaling involves normalizing or standardizing the features so that they are on a similar scale, which is essential for many machine learning algorithms. Feature selection is the process of choosing the most relevant features, which can significantly enhance the model's performance by reducing noise and focusing on the most informative aspects of the data.

3. Machine Learning Algorithm

At the core of this methodology is the machine learning algorithm, specifically a Feedforward Neural Network (FNN). This algorithm is selected for its ability to learn complex patterns in the data, making it well-suited for phishing detection tasks. The FNN, or another chosen algorithm, will be trained using the prepared data to classify URLs as either phishing or legitimate.

4. Modeling

Modeling involves the actual training of the machine learning model. The prepared features are fed into the FNN, and the model's parameters are adjusted to minimize the error between predicted and actual outcomes. This phase is where the model learns to make accurate predictions based on the data provided.

5. Model Evaluation & Tuning

Once the model is trained, it is evaluated to assess its performance. This involves using metrics such as accuracy, precision, recall, and F1 score to determine how well the model is performing. If necessary, hyperparameters are tuned to optimize the model, ensuring that it performs at its best.

6. Deployment & Monitoring

After satisfactory performance is achieved, the model is deployed in a production environment. Deployment means that the model is now being used to detect phishing attacks in real-time. Continuous monitoring is implemented to track the model's performance, ensuring that it remains effective as new phishing techniques emerge.

7. Iteration Loop

The process is iterative, meaning that it does not end with the initial deployment. Continuous improvement is built into the methodology, allowing the team to revisit data preparation, modeling, and evaluation stages as needed. This iterative loop ensures that the model evolves and adapts to the constantly changing landscape of phishing attacks.

CHAPTER 6

IMPLEMENTATION

This section details how to build a machine learning model for detecting phishing URLs. It starts by checking for GPU availability and loading the dataset. Next, it cleans the data and extracts features like URL length and directory length. The data is then split into training and testing sets, standardized, and used to build a neural network model with TensorFlow. Finally, the model is trained and evaluated to ensure it performs well in detecting phishing attempts.

6.1 Checking GPU Availability and Loading the Dataset

```
# Check if GPU is being used.
import tensorflow as tf
tf.test.gpu_device_name()
import pandas as pd
# Loading the downloaded dataset
df = pd.read_csv("/content/drive/MyDrive/urldata.csv")
df.head(10)
```

This code segment accomplishes two important tasks: verifying the availability of a GPU and loading the dataset into a Pandas DataFrame.

The first part of the code checks for GPU availability. TensorFlow, a popular deep learning library, is used for this purpose. The function `tf.test.gpu_device_name()` is called to check if TensorFlow can access a GPU. If a GPU is available, TensorFlow will utilize it to speed up computations, which is especially beneficial for training deep learning models. Training on a GPU can significantly reduce the time required compared to training on a CPU. The second part of the code loads the dataset. Pandas is used to read a CSV file containing URL data. The `pd.read_csv` function is called with the path to the dataset file. Once the data is loaded into a DataFrame named `df`, the `df.head(10)` function is used to display the first 10 rows. This provides an initial view of the dataset, allowing you to verify that the data has been loaded correctly and to get a sense of its structure. This initial inspection is crucial for planning further data preprocessing and feature engineering steps.

Combining these steps ensures that the environment is set up correctly for efficient computation and that the dataset is ready for subsequent analysis. Checking GPU availability

is a best practice in machine learning projects that involve large datasets or complex models, as it can greatly enhance performance. Loading the dataset and displaying its initial rows helps in understanding the nature of the data, which is essential for effective preprocessing and model building.

6.2 Cleaning the Dataset and Obtaining Data Information

```
#Removing the unnamed columns as it is not necessary.
df = df.drop('Unnamed: 0', axis=1)
df.head(10)
df.info()
```

This segment of code focuses on cleaning the dataset by removing unnecessary columns and obtaining detailed information about the dataset's structure.

The first part of the code removes an unnecessary column named 'Unnamed: 0'. This column might have been created during the data collection or export process, and it does not contribute to the analysis. Removing such columns is a crucial step in data cleaning, as they can introduce noise and reduce the efficiency of the analysis. The `df.drop` method is used to remove this column, with `axis=1` indicating that a column (and not a row) is to be dropped. After dropping the column, `df.head(10)` is called again to display the first 10 rows of the updated DataFrame. This helps verify that the unnecessary column has been successfully removed and ensures that the dataset is cleaner and more manageable.

The second part of the code uses the `df.info()` function to print a concise summary of the DataFrame. This summary includes the number of entries (rows), the column names, the number of non-null entries for each column, and the data types of each column. This information is essential for understanding the structure of the dataset. For example, it can help identify columns with missing values, which may require further cleaning, or determine whether certain columns need to be converted to different data types to facilitate analysis.

Understanding the dataset's structure through `df.info()` provides a foundation for effective data preprocessing and feature engineering. It helps in making informed decisions about handling missing values, encoding categorical variables, and normalizing numerical features. Moreover, this summary can reveal potential issues with the data that might need to be addressed before proceeding to more complex analysis and modeling steps.

6.3 Understanding Dataset Shape and Label Distribution

This segment of the code focuses on understanding the overall structure of the dataset and analyzing the distribution of the target labels.

The first line, `df.shape`, returns the shape of the DataFrame, which includes the number of rows and columns. This information is critical as it provides a sense of the dataset's size. A large number of rows indicates a substantial amount of data, which can be beneficial for training a robust model. Conversely, a large number of columns can indicate a wide range of features that may need to be carefully managed and processed. Knowing the shape of the dataset helps in planning the computational resources required and the complexity of the data processing pipeline.

The second part of the code analyzes the distribution of the target labels by calling `df["label"].value_counts()`. The `label` column typically contains binary values indicating whether a URL is legitimate or fraudulent. By counting the occurrences of each label, we can determine if the dataset is balanced or imbalanced. A balanced dataset, where the number of legitimate and fraudulent URLs is approximately equal, is ideal for training most machine learning models. An imbalanced dataset, where one class significantly outnumbers the other, can pose challenges for model training. Models trained on imbalanced data may become biased towards the majority class, leading to poor performance in detecting the minority class.

```
df.shape
# Printing number of legit and fraud domain URLs
df["label"].value_counts()
```

Understanding the label distribution is crucial for designing effective strategies to handle potential imbalances. Techniques such as resampling (oversampling the minority class or undersampling the majority class), using different performance metrics (such as precision, recall, and F1 score), or employing specialized algorithms (like SMOTE or anomaly detection) can be applied to address these issues. By knowing the label distribution upfront, we can make informed decisions about the best approach to ensure that our model performs well across all classes.

6.4 Initial Data Processing and Feature Extraction

This segment focuses on the initial steps of data processing and feature extraction, which are crucial for preparing the data for machine learning models.

First, the code imports necessary dependencies. The `urlparse` module from Python's standard library is used for parsing URLs into components (such as scheme, netloc, path, etc.), and the `os.path` module provides functions for manipulating file and directory paths. These imports set up the environment for feature extraction related to URLs.

Next, the DataFrame variable is reassigned to `urldata` for clarity and consistency in further operations. This ensures that any subsequent modifications are made to the `urldata` DataFrame.

```
#Importing dependencies
from urllib.parse import urlparse
import os.path
# changing dataframe variable
urldata = df
# Length of URL (Phishers can use long URL to hide the doubtful part in the address bar)
urldata['url_length'] = urldata['url'].apply(lambda i: len(str(i)))
# Hostname Length
urldata['hostname_length'] = urldata['url'].apply(lambda i: len(urlparse(i).netloc))
# Path Length
urldata['path_length'] = urldata['url'].apply(lambda i: len(urlparse(i).path))
```

The following lines of code create new features that capture various aspects of the URLs:

1. URL Length:

```
urldata['url_length'] = urldata['url'].apply(lambda i: len(str(i)))
```

This feature calculates the length of each URL. Phishers often use long URLs to hide malicious parts of the URL, making it harder for users to detect fraud. By calculating the length of each URL, this feature helps in identifying potentially suspicious URLs.

2. Hostname Length:

```
urldata['hostname_length'] = urldata['url'].apply(lambda i: len(urlparse(i).netloc))
```

This feature calculates the length of the hostname part of the URL. The hostname can provide valuable insights

3. Path Length:

```
urldata['path_length'] = urldata['url'].apply(lambda i: len(urlparse(i).path))
```

This feature calculates the length of the path component of the URL. The path often contains important information about the specific resource being accessed, and longer paths might indicate attempts to obfuscate the true nature of the URL.

Feature extraction is a critical step in the machine learning pipeline, as it transforms raw data into meaningful inputs that can be used to train models. By deriving these features, the model can better understand patterns and characteristics associated with phishing URLs, thereby improving its ability to detect them. This step sets the foundation for further feature engineering and model training, ensuring that the data is well-prepared for effective analysis.

6.5 Feature Extraction: First Directory Length

```
# First Directory Length
def fd_length(url):
    urlpath = urlparse(url).path
    try:
        return len(urlpath.split('/')[1])
    except:
        return 0
urldata['fd_length'] = urldata['url'].apply(lambda i: fd_length(i))
```

This segment continues the feature extraction process by adding another important feature: the length of the first directory in the URL path.

1. Function Definition:

This function, `fd_length`, is defined to calculate the length of the first directory in the URL path. The `urlparse` function is used to extract the path component of the URL. The path is then split by the '/' delimiter, and the length of the first directory is calculated. The try-except block ensures that the function handles any potential errors gracefully, returning a length of 0 if the URL path does not contain a first directory.

```
def fd_length(url):
    urlpath = urlparse(url).path
    try:
        return len(urlpath.split('/')[1])
    except:
        return 0
```

2. Applying the Function:

```
urldata['fd_length'] = urldata['url'].apply(lambda i: fd_length(i))
```

The function `fd_length` is applied to each URL in the DataFrame using the `apply` method. This creates a new feature, `fd_length`, which stores the length of the first directory for each URL.

The length of the first directory can provide additional insights into the structure of the URL. Phishing URLs might use unusual directory lengths to disguise their true intent or to mimic the structure of legitimate websites. By capturing this information, the model gains another useful feature that can help in distinguishing between legitimate and fraudulent URLs.

Feature extraction is a crucial step in preparing the data for machine learning. Each new feature adds a piece of information that can improve the model's ability to detect patterns and make accurate predictions. By carefully selecting and engineering features, we can enhance the model's performance and make it more robust in detecting phishing attempts.

6.6 Data Inspection

```
# Printing first few rows
urldata.head(10)
```

This segment involves inspecting the processed data by printing the first few rows of the updated DataFrame.

The `urldata.head(10)` function is called to display the first 10 rows of the DataFrame after feature extraction. This step is crucial for verifying that the new features have been correctly added and that the data is in the expected format.

Inspecting the data after each major processing step helps ensure that the operations have been performed correctly. By printing the first few rows, you can visually confirm that the new features (such as `url_length`, `hostname_length`, `path_length`, and `fd_length`) have been successfully added to the DataFrame. This visual inspection can also help identify any anomalies or errors that might have occurred during the feature extraction process.

Data inspection is an essential part of the data preprocessing workflow. It allows you to catch and address issues early in the process, ensuring that the data is well-prepared for subsequent analysis and model training. By regularly inspecting the data, you can maintain the quality and integrity of the dataset, which is critical for building reliable and accurate machine learning models.

6.7 Data Splitting and Standardization

1.Importing Libraries:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

The code imports necessary functions from the `sklearn` library. `train_test_split` is used to split the dataset into training and testing sets, and `StandardScaler` is used for standardizing the features.

2. Data Splitting:

```
X = urldata.drop('label', axis=1)
y = urldata['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

The DataFrame is split into feature variables (`X`) and the target variable (`y`). The `label` column, which indicates whether a URL is legitimate or fraudulent, is used as the target variable. The remaining columns are used as features. The `train_test_split` function is then used to split the data into training and testing sets. 80% of the data is used for training, and 20% is reserved for testing. The `random_state=42` parameter ensures that the split is reproducible.

3. Standardization:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Standardization is performed to scale the features so that they have a mean of 0 and a standard deviation of 1. This is important because features with different scales can negatively impact the performance of machine learning algorithms. The `StandardScaler` is first fitted to the training data using `fit_transform`, and then the same scaling is applied to the testing data using `transform`. This ensures that both training and testing data are scaled consistently.

Splitting the data into training and testing sets is crucial for evaluating the performance of the model. The training set is used to train the model, while the testing set is used to evaluate its performance on unseen data. Standardizing the features helps in improving the convergence of gradient-based learning algorithms and ensures that the model treats all features equally.

6.8 Building the Neural Network Model

This segment involves building the neural network model using the TensorFlow Keras library. The chosen model is a Feed Forward Neural Network (FFNN), which is a type of artificial neural network where connections between the nodes do not form a cycle.

1. Importing Libraries:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

The necessary classes for building the model are imported from the TensorFlow Keras library. `Sequential` is a linear stack of layers, and `Dense` is a fully connected layer.

2. Building the Model:

```
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

The model is built using the Sequential API. The architecture consists of three layers:

- The first layer is a Dense layer with 64 units and a ReLU activation function. The `input_dim` parameter is set to the number of features in the training data.
- The second layer is another Dense layer with 32 units and a ReLU activation function.
- The third layer is a Dense layer with 1 unit and a sigmoid activation function. This output layer is designed for binary classification, where the output is a probability score between 0 and 1.

ReLU (Rectified Linear Unit) is a popular activation function for hidden layers as it introduces non-linearity while being computationally efficient. The sigmoid activation function is used in the output layer for binary classification, as it outputs a value between 0 and 1, representing the probability of the positive class.

Building the neural network involves defining the architecture, including the number of layers, the number of units in each layer, and the activation functions. The chosen architecture aims to balance complexity and performance, providing enough capacity to learn from the data without being too prone to overfitting.

6.9 Compiling the Model

```
# Compiling the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

This segment focuses on compiling the neural network model, which is a critical step before training the model.The `compile` method is called on the model to configure the learning process.

Three key parameters are specified:

- `loss`: The loss function to be optimized during training. For binary classification tasks, `binary_crossentropy` is a common choice. It measures the performance of the model by comparing the predicted probabilities to the actual class labels.
- `optimizer`: The optimization algorithm to use. `adam` (Adaptive Moment Estimation) is chosen for this model. Adam is an efficient and popular optimizer that combines the benefits of both the AdaGrad and RMSProp algorithms, making it suitable for a wide range of problems.
- `metrics`: The metrics to be evaluated during training and testing. `accuracy` is used to measure the proportion of correctly predicted instances out of the total instances.

Compiling the model is an essential step that ties together the architecture and the learning process. The choice of loss function, optimizer, and metrics impacts how the model learns from the data and how its performance is evaluated. `binary_crossentropy` is appropriate for the binary classification task of detecting phishing URLs, as it effectively penalizes incorrect predictions and helps in learning accurate probability distributions. The Adam optimizer is chosen for its adaptive learning rate properties and computational efficiency, making it a robust choice for training deep learning models.

6.10 Training the Model

This final segment involves training the neural network model on the training data.

```
# Training the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

The `fit` method is called on the model to train it on the training data. Several parameters are specified:

- `X_train` and `y_train`: The training features and labels, respectively.
- `epochs`: The number of epochs (complete passes through the training dataset). Here, the model is trained for 50 epochs.

- `batch_size`: The number of samples per gradient update. A batch size of 32 is used, meaning the model updates its weights after processing 32 samples.
- `validation_split`: The proportion of the training data to be used as validation data. 20% of the training data is set aside for validation to monitor the model's performance on unseen data during training.

Training the model involves multiple iterations where the model learns to minimize the loss function. The training process is monitored by evaluating the model on the validation data at the end of each epoch. This helps in detecting overfitting, where the model performs well on the training data but poorly on the validation data.

The `history` object returned by the `fit` method contains details about the training process, including the loss and accuracy metrics for each epoch. This information can be used for further analysis and visualization to understand how the model's performance evolves over time.

Training the model is a crucial step where the neural network learns from the data to make accurate predictions. By specifying the number of epochs, batch size, and validation split, the training process is carefully controlled to ensure that the model converges to a good solution while avoiding overfitting.

These code segments and explanations cover the key steps involved in data preprocessing, feature extraction, model building, compilation, and training. Each step is essential for building a robust and accurate machine-learning model for detecting phishing URLs.

CHAPTER 7

RESULTS

7.1 Datasets

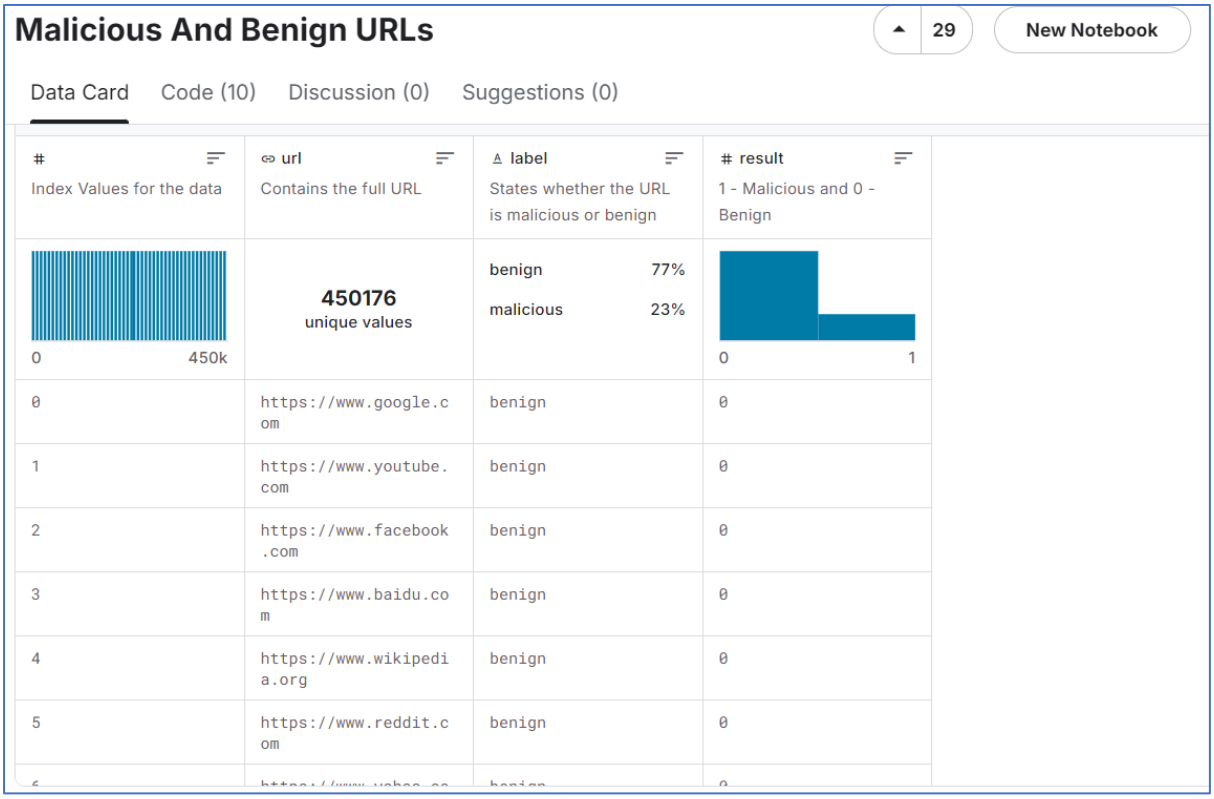

Fig 7.1 Dataset used for training

As shows in Fig 7.1, the dataset consists of 450,176 domain URLs, with 77% (approximately 345,136) labeled as legitimate (benign) and 23% (around 104,040) labeled as malicious. It is used for training machine learning models to detect phishing and malicious URLs. For this project, a subset of 10,000 URLs from each class (benign and malicious) was randomly selected to train the models effectively.

7. 2 Analysis

This section provides a comprehensive analysis of various aspects of the URL classification system, including feature correlations, model performance over time, and a comparison of different classifiers. The analysis begins with a heatmap illustrating the correlations between features extracted from URLs, highlighting relationships that can guide feature selection for machine learning models. It then evaluates the training and testing accuracy and loss across epochs, demonstrating the model's learning effectiveness. Further, it compares the performance of different classifiers on the KPT-12 dataset, focusing on accuracy and F1 scores.

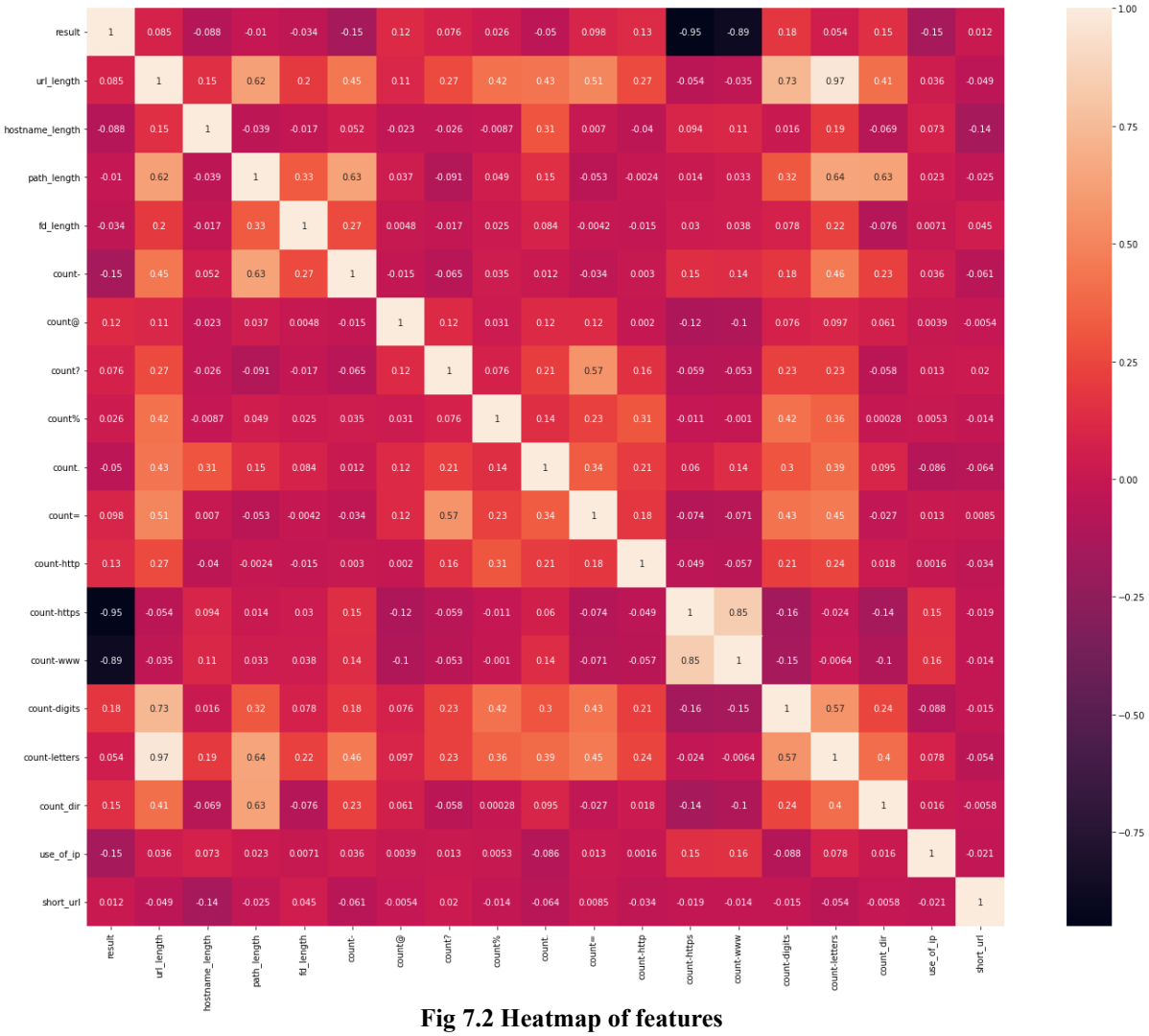

Fig 7.2 Heatmap of features

This heatmap illustrates the correlations between 16 features extracted from URLs in your dataset, with each cell representing the correlation between two features:

- **Color Scale**: The colors range from dark purple (indicating a strong negative correlation, close to -1) to bright yellow (indicating a strong positive correlation, close to +1). Lighter colors indicate weaker correlations.
- **Positive Correlation**: Features that increase or decrease together have a positive correlation, shown in yellow. For example, features like count-www and count-https are positively correlated, meaning that URLs with "www" are likely to also contain "https."
- **Negative Correlation**: Features that have an inverse relationship are negatively correlated, shown in purple. For instance, the feature result (indicating whether a URL is malicious or benign) might negatively correlate with count-digits, suggesting that malicious URLs might have fewer digits.

- **Neutral/Weak Correlation**: Features with little to no correlation are closer to 0 and are represented by colors in the middle of the scale.

This map helps identify which features are strongly related and can be useful for selecting the most relevant features for your machine-learning model.

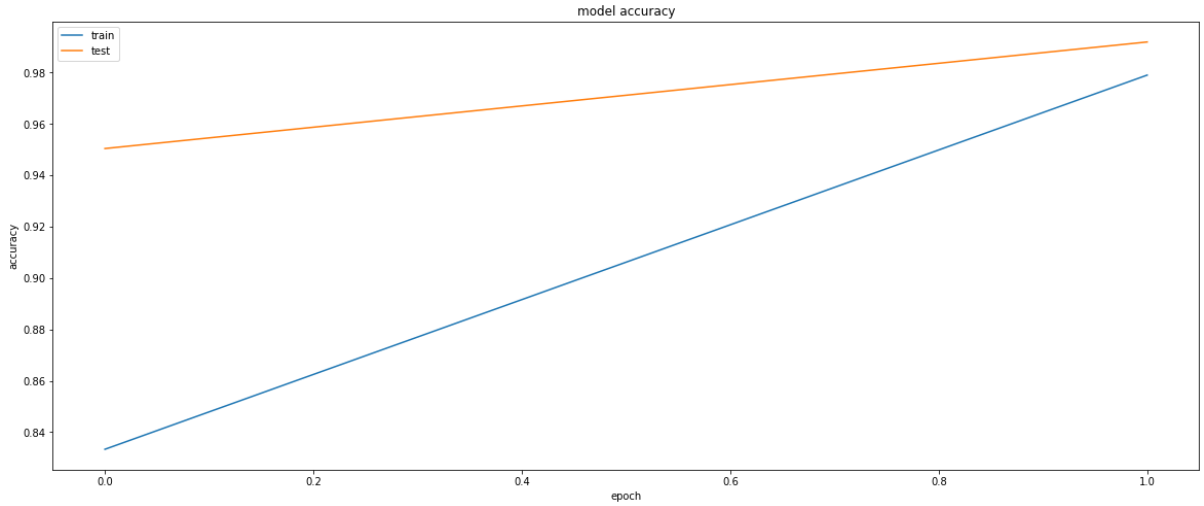

Fig 7.3 Graph of model accuracy

This graph shows the accuracy of a machine learning model over time, plotted across epochs (iterations of training):

- **Blue Line (Train Accuracy)**: Represents the accuracy of the model on the training dataset. It starts lower but gradually increases as the model learns from the data, improving its ability to make correct predictions.
- **Orange Line (Test Accuracy)**: Represents the accuracy of the model on the test dataset, which is used to evaluate how well the model generalizes to new, unseen data. The test accuracy starts high and improves slightly over time.
- **Comparison**: The test accuracy is consistently higher than the training accuracy, which says that the model is generalizing well without overfitting.

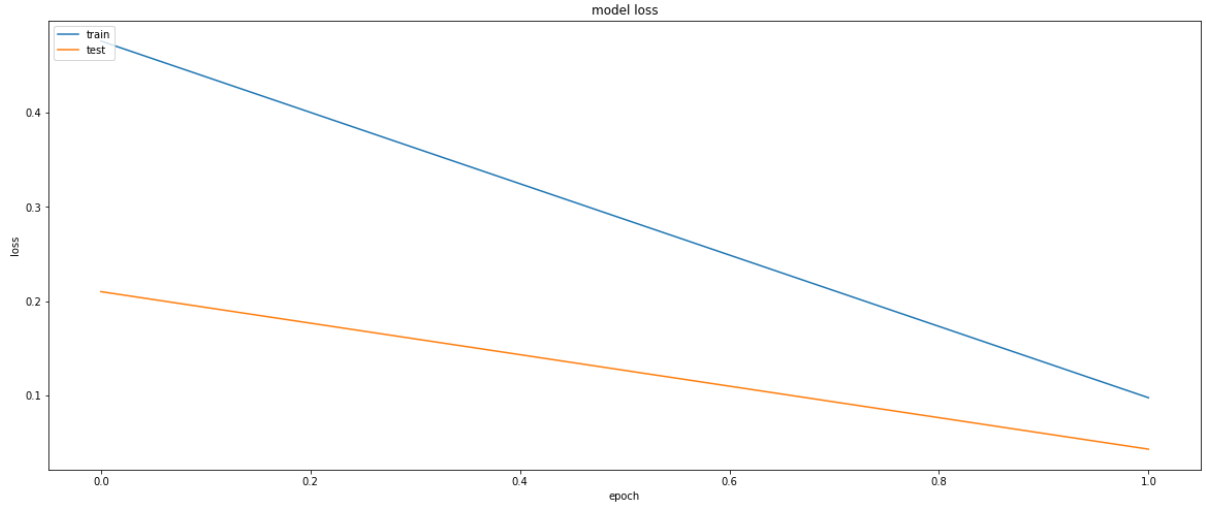

Fig 7.4 Graph of model loss

The graph in Fig 7.4 shows the loss values during the training and testing phases of the model over a series of epochs. The blue line represents the training loss, while the orange line represents the testing loss. Both lines indicate a decreasing trend, suggesting that the model is learning effectively. The decreasing loss in both the training and test datasets is a positive sign, indicating that the model's predictions are becoming more accurate as training progresses.

Analysis of previous works

The first of these datasets: Legitimate sites from Alexa database and phishing sites from PhishTank. Second: Legitimate sites from common-crawl and phishing sites from PhishTank. Third: Legitimate sites from both common-crawl and Alexa databases whereas phishing sites from PhishTank. The numbers of URLs in these datasets are given in Table 7.1. [3]

DATASETS

	Dataset-1	Dataset-2	Dataset-3
Phishing	40,668	40,668	40,668
Not Phishing	43,189	42,220	85,409
Total	83,857	82,888	126,077

Table 7.1 Various datasets

TEST RESULTS OF CLASSIFIERS ON DATASET-1

Classifier	Accuracy (%)	Time(sec.)
XGBOOST	92.95	622.6
RF	**94.59**	784.3
LR	91.31	20.5
KNN	91.49	413.7
SVM	87.03	7537.4
DT	92.59	18.8
ANN	94.35	57.6
NB	88.35	3.9

Table 7.2 Test result on dataset 1

Table 7.2 contains the accuracy rate and training period for Dataset-1. According to these data, the highest test classification result was obtained in the model using RF classifier with an accuracy rate of 94.59%. At the same time, the NB algorithm has the shortest training time.

TEST RESULTS OF CLASSIFIERS ON DATASET-2

Classifier	Accuracy (%)	Time(sec.)
XGBOOST	83.69	395.6
RF	**90.5**	439.1
LR	75.65	51.7
KNN	81.47	154.5
SVM	70.2	9833.6
DT	81.67	23
ANN	88.22	40.3
NB	70.05	1.8

Table 7.3 Test result on dataset 2

Table 7.3 contains the accuracy rate and training time taken for Dataset-2. According to these data, the highest test classification result was obtained in the model using RF classifier with an accuracy rate of 90.50%. At the same time, the NB algorithm has the shortest training time.

TEST RESULTS OF CLASSIFIERS ON DATASET-3

Classifier	Accuracy (%)	Time(sec.)
XGBOOST	83.27	271.4
RF	**91.26**	133.2
LR	78.26	63.5
KNN	81.11	262.1
SVM	76.76	15956.2
DT	81.66	17.8
ANN	88.88	29.4
NB	67.04	2.1

Table 7.4 Test result on dataset 3

Table 7.4 contains the accuracy rate and training time taken for Dataset-3. According to these data, the highest test classification result was obtained in the model using RF classifier with an accuracy rate of 91.26%. At the same time, the NB algorithm has the shortest training time.

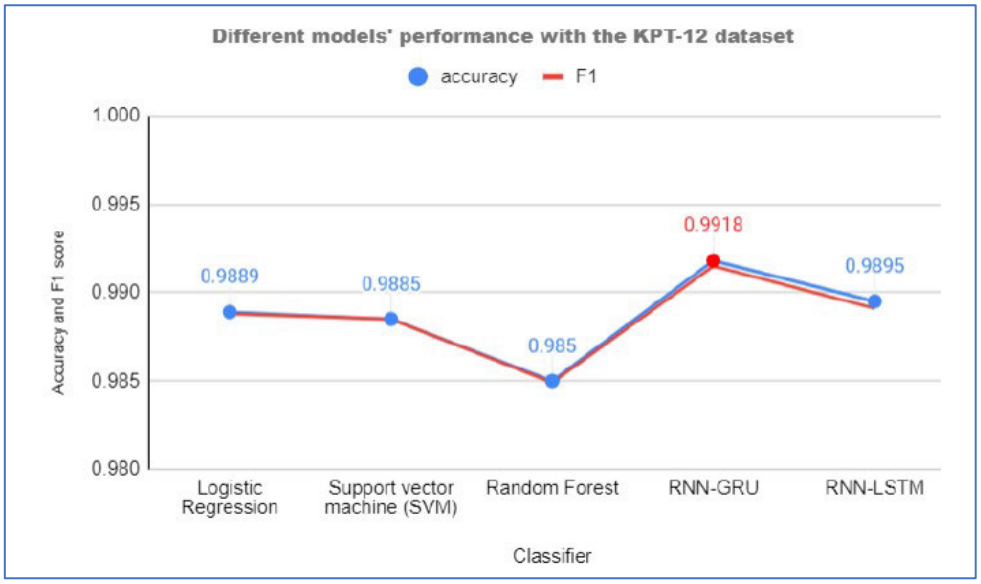

Fig 7.5 Graph of different models accuracy

The graph in Fig 7.5 illustrates the performance of various classifiers on the KPT-12 dataset, comparing their accuracy (blue points) and F1 scores (red points). The classifiers evaluated include Logistic Regression, Support Vector Machine (SVM), Random Forest, RNN-GRU, and RNN-LSTM.

- **Logistic Regression** achieved an accuracy of 0.9889 and an F1 score close to this value.
- **SVM** performed similarly with an accuracy of 0.9885.
- **Random Forest** showed a slight decrease with an accuracy of 0.985.
- **RNN-GRU** outperformed the other models, reaching the highest accuracy of 0.9918.
- **RNN-LSTM** followed closely with an accuracy of 0.9895.

The graph highlights that RNN-GRU has the best performance on this dataset, based on both accuracy and F1 score.[2]

SNAPSHOTS

The following figures give a visual overview of the main features and outputs of the URL classification system created in this project. Starting with the home page (Fig 7.6), users can easily navigate and interact with the system. The next figures show how the system tests both genuine and potentially harmful URLs (Fig 7.7 and Fig 7.8), checks the history of URL lookups (Fig 7.9), and displays real-time terminal output as the system runs (Fig 7.10). These snapshots highlight how the system works to check and verify URLs to improve web security.

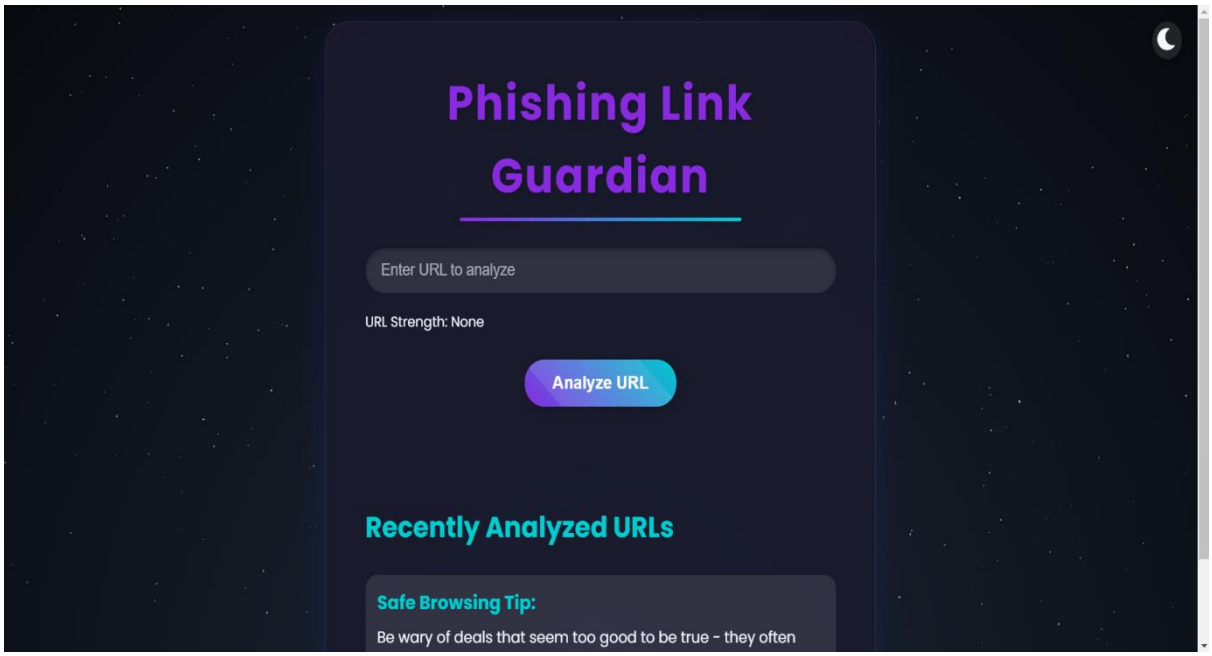

Fig 7.6 Home page of website

Fig 7.6 shows the home page of the application, providing an overview of its features and functionalities. It serves as the starting point for users to interact with the system.

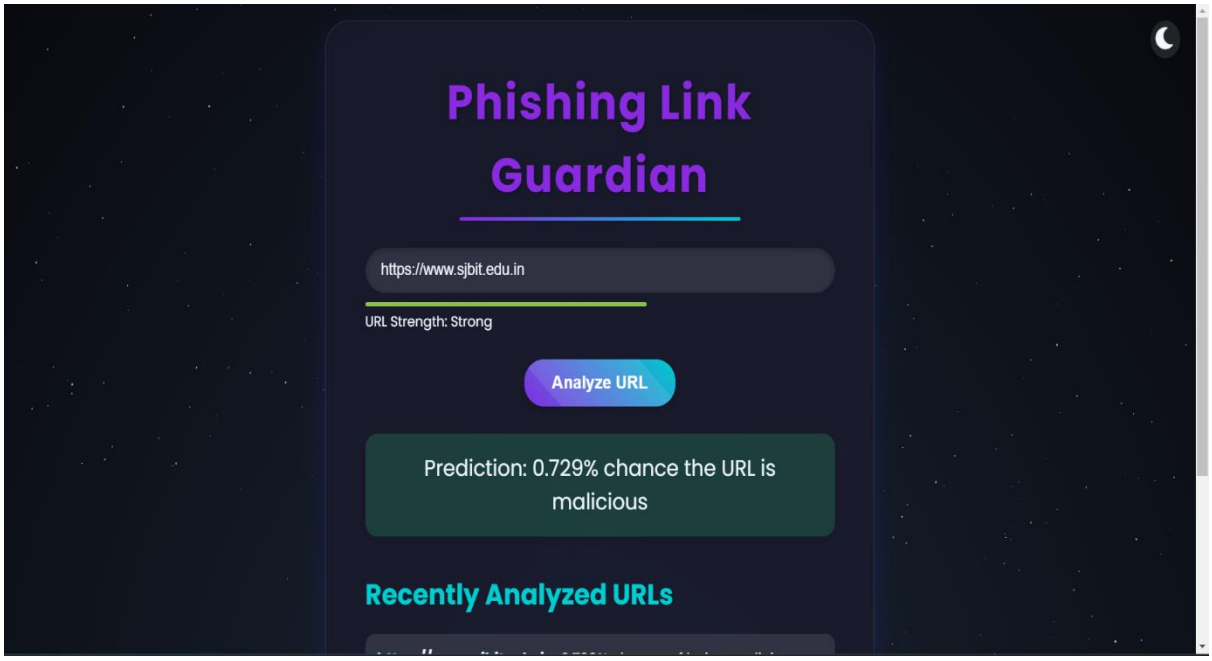

Fig 7.7Testing genuine website

Fig 7.7 illustrates the testing of a genuine website URL, demonstrating how the system verifies and confirms the authenticity of legitimate websites.

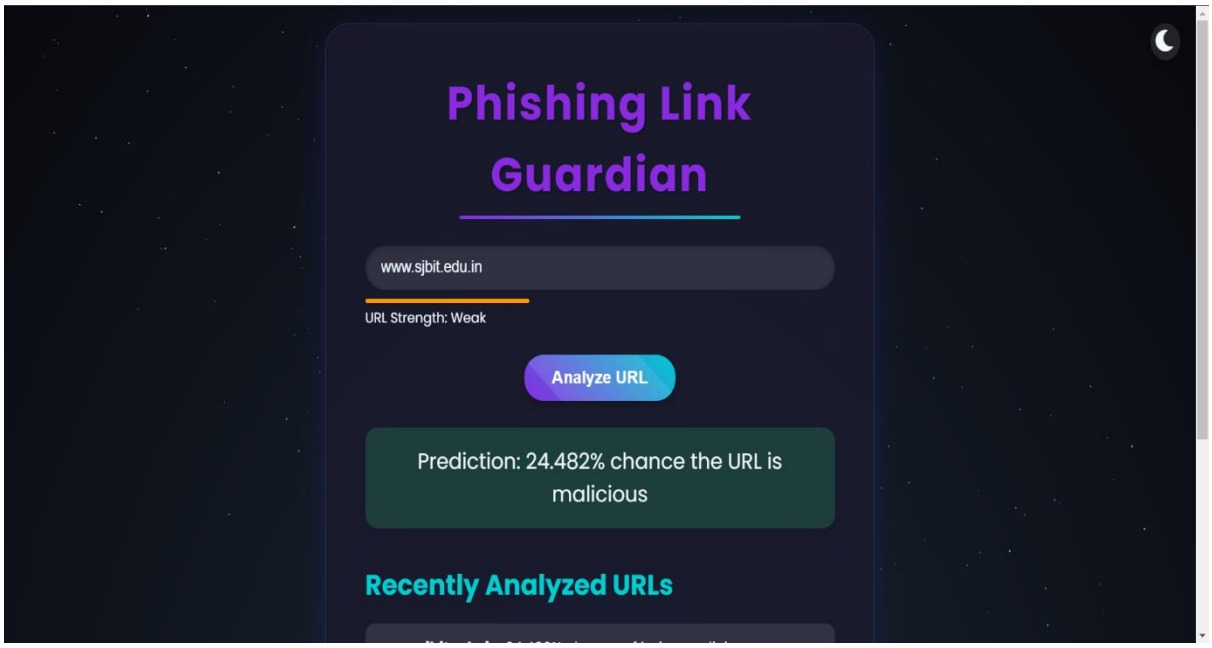

Fig 7.8 Testing Edited URL

Fig 7.8 depicts the process of testing an edited or potentially malicious URL. It shows how the system identifies, and flags altered URLs that could pose security risks.

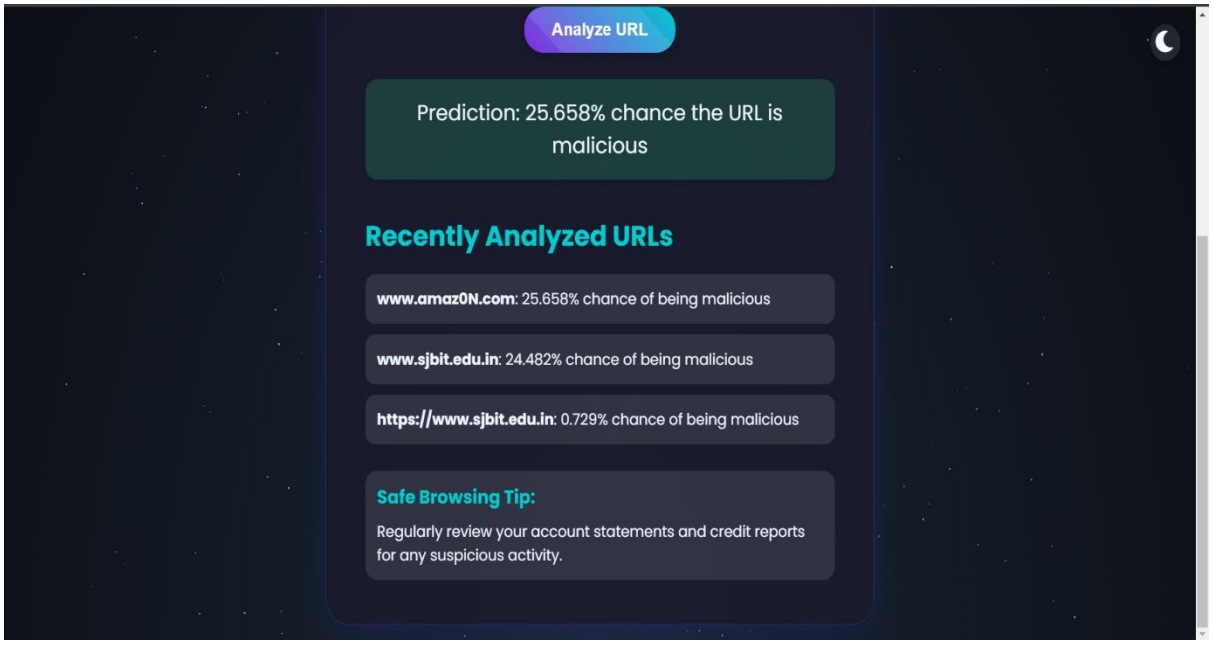

Fig 7.9 URL lookup History

Fig 7.9 presents the URL lookup history, displaying a log of previously tested URLs. It allows users to review past activities and the results of URL verifications.

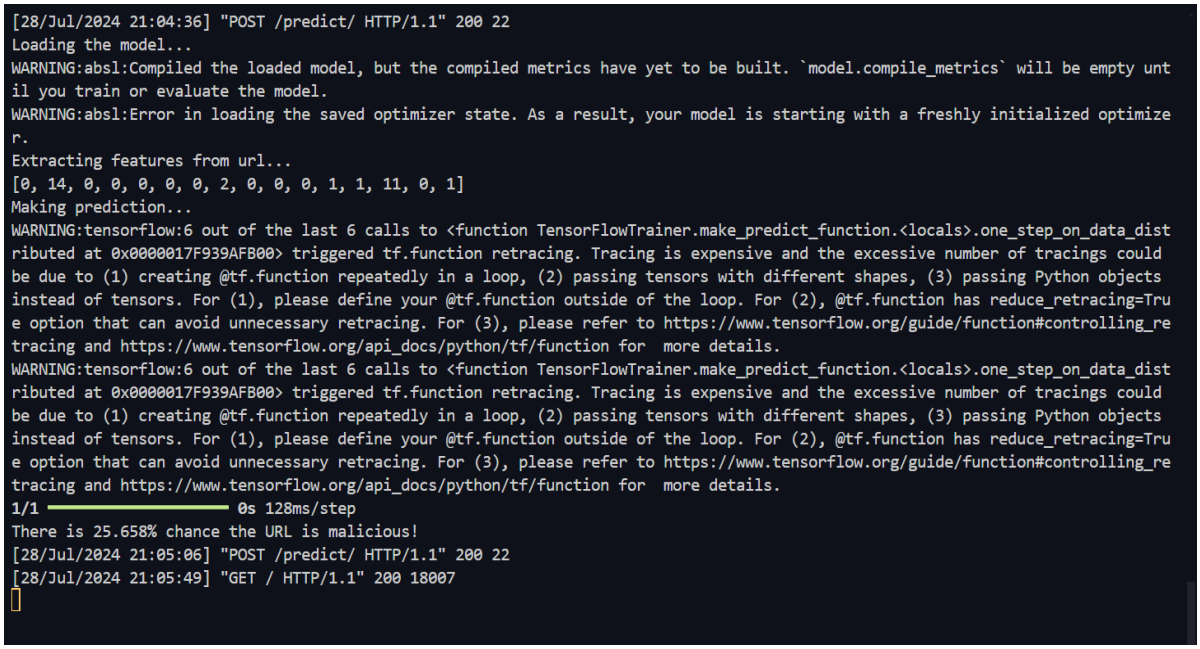

Fig 7.10 Terminal Output

Fig 7.10 shows the terminal output of the system during its operation. It provides a real-time view of the system's internal processes and the results of URL tests.

CONCLUSION

In this project, a strong system was developed to detect malicious links and prevent phishing attacks using a Feed Forward Neural Network (FFNN). The work involved several important steps, such as data retrieval, preprocessing, feature extraction, model building, and evaluation. Each step was carefully managed to make sure the model could effectively tell the difference between real and fake URLs.

A diverse set of URLs was collected and prepared as part of the dataset. Through detailed feature engineering, meaningful attributes were extracted, which helped in identifying patterns related to phishing attacks. The neural network model was built using the TensorFlow Keras library, with a multi-layered structure. ReLU activation functions were used for hidden layers, and a sigmoid function was applied in the output layer to support efficient learning and binary classification. The model was compiled with the binary cross-entropy loss function and the Adam optimizer, ensuring adaptive and effective learning.

The system developed in this project was able to overcome the limitations of existing solutions, addressing issues such as limited feature sets, poor generalization, and high computational overhead. By including detailed feature extraction and preprocessing steps, the model's performance was further improved.

Through rigorous training and validation, significant accuracy and robustness in detecting phishing URLs were demonstrated by the model. The standardized workflow, along with advanced neural network techniques, ensures that the system can be effectively used in real-world situations.

In conclusion, this project has provided a comprehensive solution to the problem of phishing attacks and has set a foundation for further improvements and innovations in cybersecurity. The methodologies and insights gained from this work can be applied to other areas, helping to enhance online security and trust.

REFERENCES

[1] Albahadili, A.J.S., Akbas, A., Rahebi, J., "Detection of phishing URLs with deep learning based on GAN-CNN-LSTM network and swarm intelligence algorithms," Springer, vol. 18, Page no. 4979—4995, 2024, doi: 10.1007/s11760-024-03204-2.

[2] Tang, L., Mahmoud, Q.H., "A Deep Learning-Based Framework for Phishing Website Detection," IEEE Access, vol. 10, issue N/A, page no 1509 – 1521, December 23, 2021, doi: 10.1109/ACCESS2021.3137636.

[3] Korkmaz, M., Sahingoz, O.K., Din, B., "Detection of Phishing Websites by Using Machine Learning-Based URL Analysis," Proceedings of the 11th ICCCNT, July 1-3, 2020, IIT Kharagpur, India, IEEE, Paper No. 49239

[4]Johnson, J., "Global Digital Population 2020," Statista, 2020, doi: https://www.statista.com/statistics/617136/digital-population-worldwide.

[5] Federal Bureau of Investigation, "2020 Internet Crime Report," Federal Bureau of Investigation,2020, doi: https://www.ic3.gov/Media/PDF/AnnualReport/2020_IC3Report.pdf.

[6] Alsariera, Y.A., Adeyemo, V.E., Balogun, A.O., Alazzawi, A.K., "AI Meta-Learners and Extra-Trees Algorithm for the Detection of Phishing Websites," IEEE Access, vol. 2020, Page no 142532 - 14254, August 3, 2020, doi: 10.1109/ACCESS.2020.3013699.

[7] Yang, P., Zhao, G., Zeng, P., "Phishing Website Detection Based on Multidimensional Features Driven by Deep Learning," IEEE Access, vol. 7, pp. 18914-18924, Jan. 2019, doi: 10.1109/ACCESS.2019.2892066.

[8] Sahingoz, O.K., Buber, E., Demir, O., Diri, B., "Machine learning based phishing detection from URLs," Expert Systems With Applications, vol. 117, pp. 345-357, 2019.

[9] Shahrivari, V., Darabi, M.M., Izadi, M., "Phishing Detection Using Machine Learning Techniques," arXiv preprint arXiv:2009.11116, Sep. 2020.

[10] Yi, P., Guan, Y., Zou, F., Yao, Y., Wang, W., Zhu, T., "WebPhishing Detection Using a Deep Learning Framework," IEEE Transactions on Information Forensics and Security, vol. 14, pp. 2975-2987, Dec. 2019, doi: 10.1109/TIFS.2019.2944422.

[11] Ahmad, S.N.W., et al., "Comparative Performance of Machine Learning Methods for Classification on Phishing Attack Detection," International Journal of Advanced Trends in Computer Science and Engineering, vol. 9, no. 1.5, pp. 349-354, 2020, doi: 10.30534/ijatcse/2020/4991.52020.

[12] Do, N.Q., Herrera-Viedma, E., Selamat, A., Krejcar, O., Fujita, H., "Deep Learning for Phishing Detection: Taxonomy, Current Challenges and Future Directions," IEEE Access, vol. 10, pp. 35388-35404, 2022, doi: 10.1109/ACCESS.2022.3151903.

[13] Salahdine, F., El Mrabet, Z., Kaabouch, N., "Phishing Attacks Detection: A Machine Learning-Based Approach," IEEE Transactions on Information Forensics and Security, vol. 16, pp. 1824-1835, 2021, doi: 10.1109/TIFS.2021.3053162.

[14] El Aassal, A., Baki, S., Das, A., Verma, R.M., "An In-Depth Benchmarking and Evaluation of Phishing Detection Research for Security Needs," IEEE Access, vol. 10, pp. 2969780, 2020, doi: 10.1109/ACCESS.2020.2969780.

[15] Harun, N.Z., Jaffar, N., Kassim, P.S.J., "Physical attributes significant in preserving the social sustainability of the traditional Malay settlement," in *Reframing the Vernacular: Politics, Semiotics, and Representation*, Springer, 2020, pp. 225–238.

[16] Alsariera, Y.A., Adeyemo, V.E., "AI Meta-Learners and Extra-Trees Algorithm for the Detection of Phishing Websites," IEEE Access, vol. 2020, pp. N/A, August 3, 2020, doi: 10.1109/ACCESS.2020.3013699.