**“Lightweight Phishing URL Detection Using Feature-Optimized XGBoost for Real-Time Classification”**

**ABSTRACT:**

Phishing attacks remain a major cybersecurity concern due to their ability to mimic legitimate websites and deceive users into revealing sensitive information. While several approaches in phishing detection have focused on heuristic rules or deep learning models, many of these methods lack transparency in feature contribution and are computationally expensive for real-time applications. This study addresses this limitation by presenting a descriptive analysis and classification of phishing URLs using a feature-driven machine learning framework. Unlike traditional detection mechanisms that rely on predefined rule sets, this research employs the XGBoost algorithm to evaluate and classify URL characteristics derived from structural, lexical, and host-based attributes. Through systematic preprocessing, feature handling, and hyperparameter tuning, the model identifies key discriminative features that influence phishing behavior across diverse URL samples. The dataset used in this study consists of labeled phishing and legitimate URLs, ensuring that the evaluation remains domain-specific and practically relevant. Experimental results demonstrate that the proposed XGBoost model produces highly accurate classifications with minimal error, validating its effectiveness for phishing detection tasks. The findings indicate that feature-optimized boosting models can support reliable, data-driven security decisions for analysts, developers, and cybersecurity practitioners. This approach not only enhances efficiency in phishing identification but also contributes to greater transparency and interpretability in automated threat detection systems.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT**  **LIST OF FIGURES**  **LIST OF SYMBOLS** | i  v  vii |
| 1. | **CHAPTER 1 : INTRODUCTION**   * 1. GENERAL   2. SCOPE OF THE PROJECT   3. OBJECTIVE   1.4 EXISTING SYSTEM  1.4.1 EXISTINGSYSTEM DISADVANTAGES  1.5 LITERATURE SURVEY  1.6 PROPOSED SYSTEM  1.6.1 PROPOSED SYSTEM ADVANTAGES |  |
| 2. | **CHAPTER 2 :PROJECT DESCRIPTION**  2.1 GENERAL  2.2 METHODOLOGIES  2.2.1 MODULES NAME  2.2.2 MODULES EXPLANATION  2.3 TECHNIQUE OR ALGORITHM |  |
| 3. | **CHAPTER 3 : REQUIREMENTS**  3.1 General  3.2 Hardware REQUIREMENTS  3.3 Software REQUIREMENTS |  |
| 4. | **CHAPTER 4 :SYSTEM DESIGN**  **4.1 general**  **4.2 uml diagrams**  4.2.1 USE CASE DIAGRAM  4.2.2 CLASS DIAGRAM  4.2.3 OBJECT DIAGRAM  4.2.4 STATE DIAGRAM  4.2.5 activity diagram  4.2.6 SEQUENCE DIAGRAM  4.2.7 COLLABORATION DIAGRAM  4.2.8 COMPONENT DIAGRAM  4.2.9 DATA FLOW DIAGRAM  4.2.10 DEPLOYMENT DIAGRAM  4.2.11 SYSTEM ARCHITECTURE |  |
| 5. | **CHAPTER 5 : DEVELOPMENT TOOLS**  5.1 general  5.2 History of Python  5.3 Importance of Python  5.4 Features of Python  5.5 Libraries used in python |  |
| 6. | **CHAPTER 6 :IMPLEMENTATION**  6.1 GENERAL  6.2 IMPLEMENTATION |  |

|  |  |  |
| --- | --- | --- |
| 7. | **CHAPTER 7 :SNAPSHOTS**  7.1 GENERAL  7.2 VARIOUS SNAPSHOTS |  |
| 8. | **CHAPTER 8 :SOFTWARE TESTING**  8.1 GENERAL  8.2 DEVELOPING METHODOLOGIES  8.3 TYPES OF TESTING |  |
| 9. | **CHAPTER 9 :**  **FUTURE ENHANCEMENT**  9.1 FUTURE ENHANCEMENTS |  |
| **10** | **CHAPTER 10 :**  10.1CONCLUSION  10.2 REFERENCES |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **NAME OF THE FIGURE** | **PAGE NO.** |
|  |  |  |
| 4.1 | Use case Diagram |  |
| 4.2 | Class diagram |  |
| 4.3 | Object diagram |  |
| 4.4 | State Diagram |  |
| 4.5 | Activity Diagram |  |
| 4.6 | Sequence diagram |  |
| 4.7 | Collaboration diagram |  |
| 4.8 | Component Diagram |  |
| 4.9 | Data flow diagram |  |
| 4.10 | Deployment Diagram |  |
| 4.11 | Architecture Diagram |  |

**LIST OF SYMBOLS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NOTATION**  **NAME** | **NOTATION** | **DESCRIPTION** |
| 1. | Class | *Class Name*  *-attribute*  *-attribute*  *+operation*  *+operation*  *+operation*  *+ public*  *-private*  *# protected* | Represents a collection of similar entities grouped together. |
| 2. | Association | name  Class B  Class A  Class A  Class B | Associations represents static relationships between classes. Roles represents the way the two classes see each other. |
| 3. | Actor | Class A  Class A  Class B  Class B | It aggregates several classes into a single classes. |
| 4. | Aggregation | Interaction between the system and external environment |

|  |  |  |  |
| --- | --- | --- | --- |
| 5. | Relation  (uses) | uses | Used for additional process communication. |
| 6. | Relation  (extends) | extends | Extends relationship is used when one use case is similar to another use case but does a bit more. |
| 7. | Communication |  | Communication between various use cases. |
| 8. | State | State | State of the processes. |
| 9. | Initial State |  | Initial state of the object |
| 10. | Final state |  | Final state of the object |
| 11. | Control flow |  | Represents various control flow between the states. |
| 12. | Decision box |  | Represents decision making process from a constraint |
| 13. | Use case |  | Interact ion between the system and external environment. |

|  |  |  |  |
| --- | --- | --- | --- |
| 14. | Component |  | Represents physical modules which are a collection of components. |
| 15. | Node |  | Represents physical modules which are a collection of components. |
| 16. | Data Process/State |  | A circle in DFD represents a state or process which has been triggered due to some event or action. |
| 17. | External entity |  | Represents external entities such as keyboard, sensors, etc. |
| 18. | Transition |  | Represents communication that occurs between processes. |
| 19. | Object Lifeline |  | Represents the vertical dimensions that the object communications. |
| 20. | Message | Message | Represents the message exchanged. |

**CHAPTER-1**

**INTRODUCTION**

In recent years, the rapid growth of digital services and online platforms has significantly increased user reliance on web-based systems for financial transactions, communication, and information exchange. However, this expansion has also led to a substantial rise in cyber threats, with phishing emerging as one of the most widespread and harmful forms of attack. Phishing websites imitate legitimate domains to deceive users into revealing sensitive information such as passwords, banking credentials, and personal identity details. Traditional detection methods, including blacklist monitoring and rule-based systems, often struggle to keep pace with evolving phishing techniques, as they fail to recognize newly generated or dynamically modified malicious URLs. This limitation highlights the pressing need for intelligent and adaptive detection approaches capable of analyzing complex URL characteristics in real time. With the increasing availability of structured phishing datasets containing lexical, structural, and domain-based attributes, machine learning has become a promising solution for developing automated and objective phishing detection frameworks. Among the wide range of machine learning algorithms, the XGBoost classifier stands out due to its ability to effectively handle high-dimensional tabular data and deliver strong generalization performance with minimal computational cost. By applying XGBoost to phishing URL datasets, meaningful feature patterns such as domain behavior, URL composition, and security certificate indicators can be identified and used to distinguish phishing websites from legitimate ones. Through systematic preprocessing, feature analysis, and hyperparameter tuning, the model is able to produce accurate classifications while maintaining transparency in feature contribution and operational efficiency. This approach not only enhances the reliability of phishing detection but also supports data-driven decision-making for cybersecurity analysts, developers, and organizations. Ultimately, the adoption of feature-optimized machine learning models contributes to more robust, scalable, and interpretable phishing prevention systems, strengthening overall security resilience against evolving cyber threats.

**1.2 SCOPE OF THE PROJECT**

The scope of this project focuses on developing a feature-based phishing URL detection model using the XGBoost algorithm to classify websites as legitimate or phishing based on structural, lexical, and domain-level attributes. The study is limited to supervised machine learning classification and operates on a labeled phishing dataset, where preprocessing, feature handling, and hyperparameter tuning are applied to improve model performance. The project emphasizes lightweight execution and computational efficiency, making the model suitable for real-time applications such as browser extensions, email filters, and web-security tools. The analysis is restricted to URL-based features and does not include content-level, user-behavior, or network-traffic-based phishing indicators. The evaluation of the model is carried out using standard performance metrics such as accuracy, precision, recall, F1-score, and confusion matrix analysis. Overall, the project aims to provide a transparent, interpretable, and deployment-ready phishing detection framework that supports data-driven cybersecurity decision-making.

**1.3 OBJECTIVE**

The objective of this project is to develop an efficient and reliable phishing URL detection model using the XGBoost algorithm to accurately classify websites as phishing or legitimate. It aims to preprocess and analyze URL-based lexical, structural, and domain features to identify the most influential attributes contributing to phishing behavior. Another objective is to optimize the model through hyperparameter tuning to achieve high accuracy while maintaining computational efficiency suitable for real-time applications. The project seeks to enhance transparency in phishing detection by adopting a feature-driven, interpretable machine learning approach. It also aims to evaluate model performance using metrics such as accuracy, precision, recall, F1-score, and confusion matrix analysis. Additionally, the project intends to design a lightweight detection framework that can be integrated into browser extensions or web-security environments. A further objective is to support data-driven decision-making for security analysts and developers through automated threat classification. Overall, the project aspires to contribute toward improving cybersecurity resilience by minimizing phishing risks through intelligent, scalable, and practical detection solutions.

**1.4 EXISTING SYSTEM:**

In the existing system, phishing website detection is carried out using deep learning-based models such as Feedforward Neural Networks (FNN), Deep Neural Networks (DNN), Wide and Deep Models, and the TabNet architecture, which classify URLs based on lexical, structural, and domain-specific features. These models are trained on phishing datasets containing attributes like URL length, domain activity period, SSL certificate details, and directory patterns, enabling them to capture complex relationships in the data and achieve high detection accuracy. However, most of these neural network architectures operate as black-box systems, where the internal decision process and feature influence remain difficult to interpret. Although TabNet introduces an attention-based feature mechanism, its performance depends heavily on hyperparameter tuning and computational resources. Wide and Deep models increase representational capacity but also add complexity, leading to longer training time and higher risk of overfitting. Similarly, FNN and DNN require extensive parameter tuning and lack built-in feature explainability, limiting their applicability in real-time detection environments. These limitations result in reduced transparency, higher computational cost, and limited deployment feasibility in lightweight security systems. Therefore, existing phishing detection approaches still face challenges in interpretability, adaptability, and operational efficiency.

**1.4.1 EXISTINGSYSTEM DISADVANTAGES:**

* Limited Interpretability of Models
* High Computational Cost
* Risk of Overfitting on Tabular Data
* Heavy Dependence on Hyperparameter Tuning
* Reduced Practical Deployment Feasibility

**1.5** **LITERATURE SURVEY**

**Title:** What is Web Application

**Author:** A. Yadav

**Year:** 2023

**Description:** This article provides an overview of web applications, their architecture, and their functional role in modern computing environments. The author explains how web applications operate over client–server platforms, emphasizing factors such as accessibility, scalability, and deployment convenience compared to traditional desktop applications. The work highlights the evolution of web applications in domains such as e-commerce, online services, and interactive platforms, while also discussing their relevance in real-time system integration. The article further explains key components such as the front-end interface, web server, and database interaction, helping to establish a foundational understanding of how web-based systems support analytical and intelligent applications.

**Title:** An Example of FNN With One Hidden Layer

**Author:** learnopencv

**Year:** 2024.

**Description**: This work explains the architecture and functioning of a Feedforward Neural Network (FNN), particularly focusing on a single hidden-layer design. It describes how input features are propagated through hidden layers using weighted connections and activation functions to generate classification outputs. The article emphasizes the suitability of FNNs for structured and tabular data classification tasks, where patterns are learned through forward propagation without feedback loops. The study also highlights the advantages of FNNs in terms of simplicity, computational efficiency, and adaptability to various supervised learning problems, making them a strong baseline model in predictive analytics and pattern recognition applications.

**Title:** Deep Neural Network (DNN) Model Architecture

**Author:** Creative Commons Attribution-NonCommercial 4.0 (ResearchGate Source)

**Year:** 2024.

**Description:** This reference presents the structural design and functional characteristics of Deep Neural Networks (DNNs), which extend traditional FNNs through the inclusion of multiple hidden layers. The architecture enables hierarchical feature extraction, allowing the model to capture complex nonlinear relationships in high-dimensional datasets. The study explains how deeper network layers progressively transform input features into more abstract representations, improving classification accuracy and predictive performance. The work also highlights key design elements such as activation functions, optimization strategies, and backpropagation, demonstrating the capability of DNNs to support advanced learning tasks across various domains.

**Title:**  Architecture of the Wide and Deep Neural Network

**Author:**  R. D. Corin

**Year:** 2024

**Description**: This work describes the Wide and Deep Neural Network architecture, which integrates shallow learning and deep learning components within a unified framework. The wide component is designed to memorize feature interactions and categorical relationships, while the deep component enables generalized learning through multilayer neural processing. The architecture is shown to be effective in tasks requiring both memorization and pattern abstraction, improving prediction accuracy and model robustness. The study further emphasizes the applicability of the Wide and Deep model in recommendation systems and tabular classification problems, where combining feature engineering and representation learning leads to enhanced performance outcomes.

**Title:** TabNet Architecture Consisting of the Encoder for Classification

**Author**: Creative Commons Attribution 4.0 (ResearchGate Source)

**Year:** 2024**.**

**Description:** This reference explains the TabNet model architecture, which applies a sequential attention mechanism to tabular data classification. The model selectively focuses on the most informative features at each decision step, improving interpretability and feature relevance compared to conventional deep learning models. The encoder-based structure enables efficient learning by balancing performance and transparency in prediction outcomes. The work emphasizes the capability of TabNet to handle structured datasets with high dimensionality while maintaining reduced computational complexity. Its attention-driven feature selection mechanism strengthens its suitability for real-world analytical applications such as fraud detection and security-based classification systems.

**1.6** **PROPOSED SYSTEM**

In the proposed system, phishing URL detection is carried out using a feature-driven machine learning framework based on the XGBoost classifier. Unlike deep learning-based approaches used in the existing system, this model focuses on tabular URL features such as lexical structure, domain characteristics, URL length, token frequency, and host attributes, making the detection process more interpretable and computationally efficient. The dataset consisting of phishing and legitimate URLs is preprocessed through missing-value handling, normalization, and feature encoding to ensure consistent representation across samples. The XGBoost algorithm is then trained to classify URLs, where boosting iterations and hyperparameters are tuned to improve accuracy, reduce misclassification, and enhance generalization. The model identifies influential features contributing to phishing behavior, allowing better transparency in decision outcomes compared to neural network architectures. Performance is evaluated using accuracy, precision, recall, F1-score, and confusion matrix metrics to validate the reliability of the system. The proposed system is designed to be lightweight and deployment-ready, making it suitable for real-time applications such as browser extensions, email filters, and web-security platforms. Overall, this approach enables an efficient, interpretable, and scalable phishing detection mechanism that supports practical cybersecurity decision-making.

**1.6.1** **PROPOSED SYSTEM ADVANTAGES:**

* Higher Interpretability
* Low Computational Cost
* Better Generalization on Tabular Data
* Feature-Driven Detection
* Practical Deployment Feasibility

**CHAPTER 2**

**PROJECT DESCRIPTION**

**2.1 GENERAL:**

This project focuses on the development of an intelligent phishing URL detection system using a feature-based machine learning approach to enhance cybersecurity and protect users from fraudulent websites. With the increasing reliance on online platforms for banking, communication, and digital transactions, phishing attacks have become a major threat, exploiting deceptive website URLs to steal personal and financial information. Traditional blacklist and rule-based detection systems are limited in adaptability, as they fail to recognize newly generated or evolving phishing links. To address this limitation, the proposed project employs the XGBoost classifier, a powerful boosting-based machine learning algorithm designed to analyze structured, tabular URL features with high accuracy and computational efficiency. The dataset used in the project consists of labeled phishing and legitimate URLs, from which lexical, structural, and domain-based attributes are extracted and processed through data cleaning, feature encoding, and normalization techniques. The model is trained to classify URLs by learning meaningful feature patterns that distinguish phishing behavior from legitimate website characteristics. Hyperparameter tuning and performance evaluation are carried out to optimize the model, using metrics such as accuracy, precision, recall, F1-score, and confusion matrix analysis to ensure reliable prediction capability. The system is designed to be lightweight and deployment-oriented, making it suitable for integration into browser extensions, email security tools, and real-time URL filtering applications. One of the key strengths of the project is its feature-driven approach, which enhances interpretability by identifying influential attributes contributing to phishing detection, unlike many deep learning-based systems that operate as black-box models. The project contributes to the development of an efficient, scalable, and transparent phishing detection framework capable of supporting cybersecurity analysts, developers, and organizations in making data-driven security decisions. Ultimately, this work aims to reduce phishing-related risks, improve online safety, and promote the practical application of machine learning in real-world cyber defense environments.

**2.2 METHODOLOGIES**

**2.2.1MODULES NAME:**

**Modules Name:**

* Data Acquisition & Integration
* Data Preprocessing & Normalization
* Feature Engineering & Selection
* Model Development & Optimization
* Prediction Evaluation & Result Analysis
  + 1. **MODULES EXPLANATION:**

**Data Acquisition & Integration**

In this module, the acquired data is cleaned and refined to remove noise, missing values, and inconsistencies. Attributes are converted into a machine-readable format, and normalization techniques are applied to maintain uniform feature scaling. Redundant and misleading data elements are eliminated to improve dataset stability. The preprocessing workflow ensures that the dataset remains reliable and structured for learning. This module enhances data quality and prepares it for further analytical processing.

**Data Preprocessing & Normalization**

In this module, the raw data is analyzed to identify meaningful information that reflects player performance. It includes understanding relationships between attributes, eliminating irrelevant features, and highlighting key variables. Deciphering data ensures that the most critical performance indicators are extracted for deeper analysis. This step bridges the gap between raw statistics and actionable insights.

**Feature Engineering & Selection**

This module focuses on identifying and extracting meaningful features that contribute to accurate phishing detection. Important attributes are retained, while irrelevant or highly correlated features are removed to reduce complexity. Feature selection strengthens model performance by improving interpretability and reducing computational overhead. Domain-specific insights are applied to ensure only relevant characteristics are preserved. This module ensures that the most significant features are utilized for model training.

**Model Development & Optimization**

In this module, suitable machine learning and deep learning models are designed, trained, and fine-tuned using optimized hyperparameters. The selected algorithms are evaluated across multiple configurations to achieve high accuracy and robustness. Training is performed using processed datasets to ensure consistent learning patterns. Performance metrics are analyzed to identify the most efficient and reliable model. This module focuses on building a well-optimized and effective phishing detection system.

**Prediction Evaluation & Result Analysis**

This module is responsible for generating prediction outcomes and analyzing model performance. The system classifies input records as phishing or legitimate and evaluates the results using accuracy-based metrics. Graphical performance visualizations and comparative assessments are carried out to validate effectiveness. Errors, misclassifications, and improvement areas are also examined. This module ensures meaningful interpretation of results and validates the overall success of the proposed model.

**2.3 TECHNIQUE USED OR ALGORITHM USED**

**2.3.1 EXISTING TECHNIQUE:**

Traditional machine learning algorithms such as Decision Tree, Random Forest, have been widely used for sports analytics, including football performance prediction. Decision Trees are simple and interpretable models that split data into branches based on feature thresholds, allowing for easy rule-based classification. However, they tend to overfit the training data when the dataset is large or contains complex patterns. Random Forest, an ensemble of multiple decision trees, was introduced to address overfitting by averaging predictions from many trees, thereby increasing stability and accuracy. Despite their effectiveness in various classification problems, these existing algorithms exhibit certain limitations when applied to complex football performance data. For example, Decision Trees and Random Forests may fail to capture subtle nonlinear relationships between player attributes and performance. Additionally, SVMs can become computationally expensive and less efficient when dealing with large datasets or multiple classes. These methods also lack built-in capabilities for feature importance evaluation, limiting their interpretability in terms of what specific attributes contribute most to player performance. As a result, there is a need for a more powerful and accurate algorithm to improve prediction quality and extract meaningful insights from football datasets.

**2.3.2 PROPOSED TECHNIQUE USED OR ALGORITHM USED:**

Support Vector Classifier is a supervised machine learning algorithm derived from Support Vector Machines (SVM). It is used for classification tasks by finding an optimal hyperplane that best separates data points belonging to different classes. The SVC maximizes the margin between class boundaries, ensuring robust classification even in high-dimensional spaces. Different kernel functions such as linear, polynomial, and radial basis function (RBF) can be applied to capture non-linear relationships within football performance data.

In the context of football player performance analysis, XGBoost offers a significant improvement over traditional algorithms due to its robustness, high predictive power, and model interpretability. It effectively processes numerous player attributes, including match statistics and positional metrics, and determines which features contribute the most to performance ratings. This leads to more reliable and insightful predictions, closely aligning with expert evaluations. The model’s flexibility allows it to be fine-tuned for various roles and positions, making it a powerful tool for coaches, analysts, and clubs aiming to make data-driven decisions for scouting, training, and match strategies.

**CHAPTER 3**

**REQUIREMENTS ENGINEERING**

**3.1 GENERAL**

We can see from the results that on each database, the error rates are very low due to the discriminatory power of features and the regression capabilities of classifiers. Comparing the highest accuracies (corresponding to the lowest error rates) to those of previous works, our results are very competitive.

**3.2** **HARDWARE REQUIREMENTS**

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It should what the system do and not how it should be implemented.

* PROCESSOR : DUAL CORE 2 DUOS.
* RAM : 4GB DD RAM
* HARD DISK : 500 GB

**3.3** **SOFTWARE REQUIREMENTS**

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team’s progress throughout the development activity.

* Operating System : Windows 10
* Platform : Spyder3
* Programming Language : Python
* Front End : Spyder3

**3.4** **FUNCTIONAL REQUIREMENTS**

A functional requirement defines a function of a software-system or its component. A function is described as a set of inputs, the behavior, Firstly, the system is the first that achieves the standard notion of semantic security for data confidentiality in attribute-based deduplication systems by resorting to the hybrid cloud architecture.

**3.5 NON-FUNCTIONAL REQUIREMENTS**

**The major non-functional Requirements of the system are as follows**

**Usability**

The system is designed with completely automated process hence there is no or less user intervention.

**Reliability**

The system is more reliable because of the qualities that are inherited from the chosen platform python. The code built by using python is more reliable.

**Performance**

This system is developing in the high level languages and using the advanced back-end technologies it will give response to the end user on client system with in very less time.

**Supportability**

The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform, which is built into the system.

**Implementation**

The system is implemented in web environment using Jupyter notebook software. The server is used as the intellignce server and windows 10 professional is used as the platform. Interface the user interface is based on Jupyter notebook provides server system.

**CHAPTER 4**

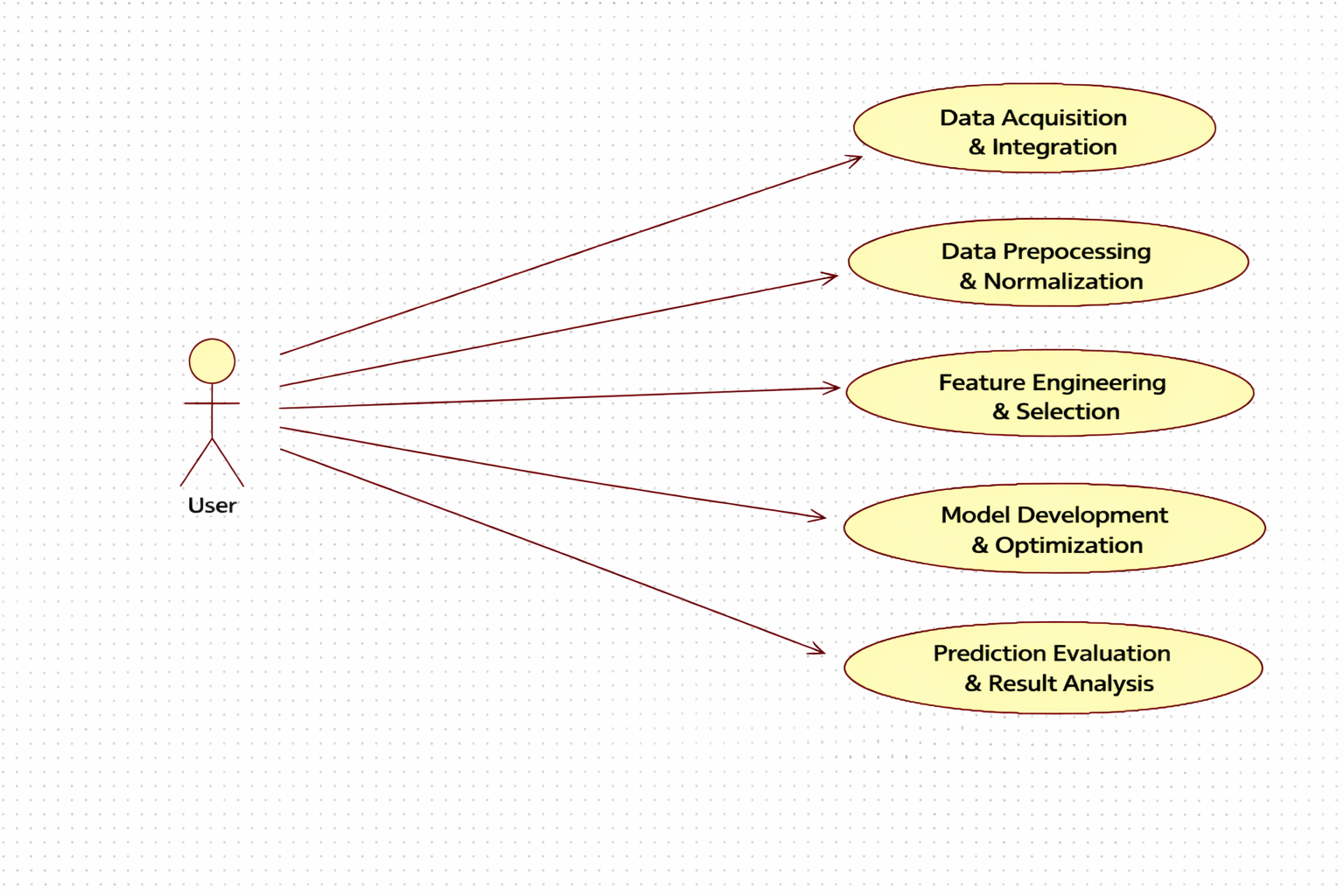
**DESIGN ENGINEERING**

**4.1 GENERAL**

Design Engineering deals with the various UML [Unified Modelling language] diagrams for the implementation of project. Design is a meaningful engineering representation of a thing that is to be built. Software design is a process through which the requirements are translated into representation of the software. Design is the place where quality is rendered in software engineering.

**4.2 UML DIAGRAMS**

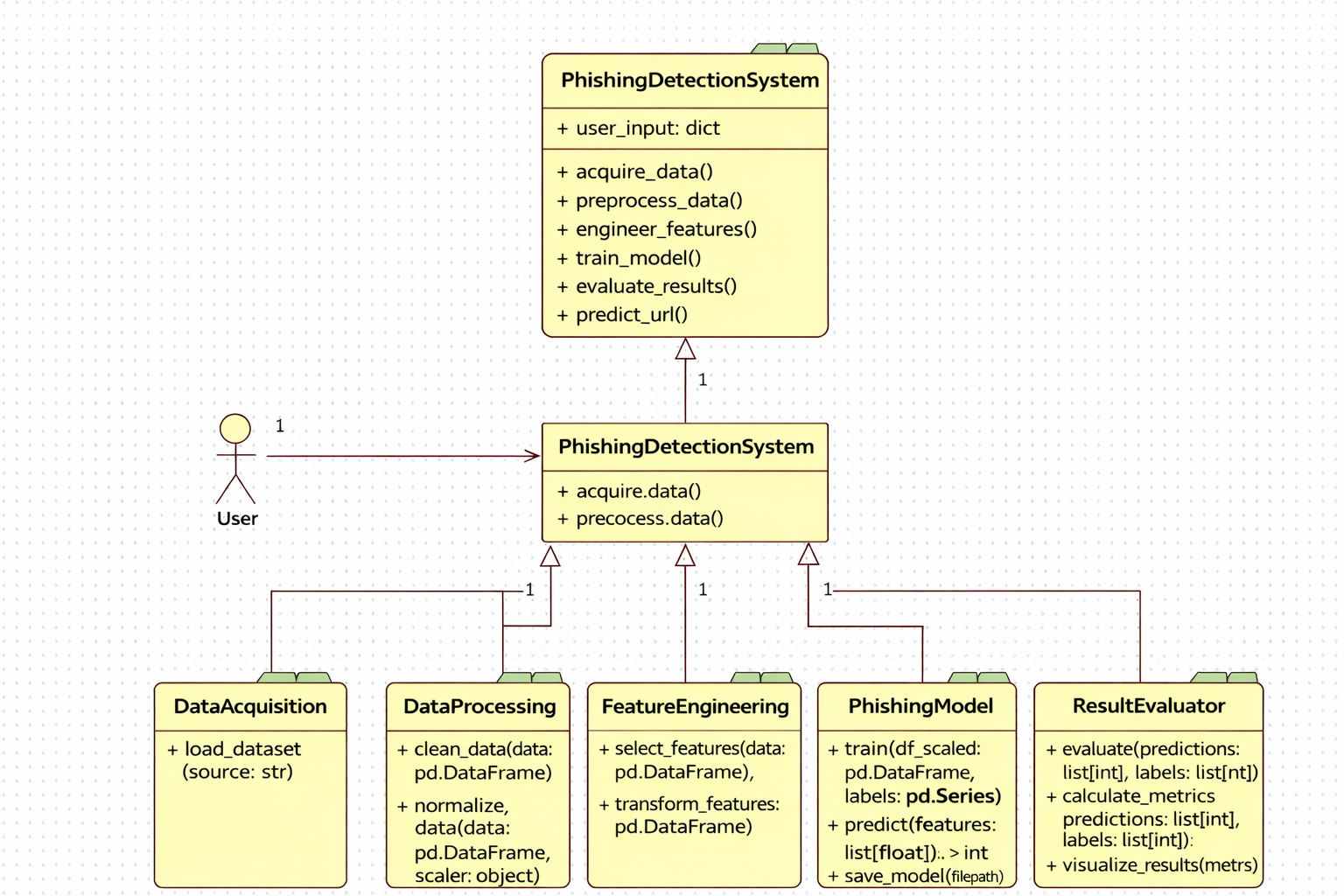
**4.2.1 USE CASE DIAGRAM**

****

**EXPLANATION:**

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. The above diagram consists of user as actor. Each will play a certain role to achieve the concept.

**4.2.2 CLASS DIAGRAM**



**EXPLANATION**

In this class diagram represents how the classes with attributes and methods are linked together to perform the verification with security. From the above diagram shown the various classes involved in our project.

**4.2.3 OBJECT DIAGRAM**

****

**EXPLANATION:**

In the above digram tells about the flow of objects between the classes. It is a diagram that shows a complete or partial view of the structure of a modeled system. In this object diagram represents how the classes with attributes and methods are linked together to perform the verification with security.

**4.2.4 STATE DIAGRAM**

****

**EXPLANATION:**

State diagram are a loosely defined diagram to show workflows of stepwise activities and actions, with support for choice, iteration and concurrency. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.

**4.2.5 ACTIVITY DIAGRAM**



**EXPLANATION:**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

**4.2.6 SEQUENCE DIAGRAM**

****

**EXPLANATION:**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

**4.2.7 COLLABORATION DIAGRAM**

****

**EXPLANATION:**

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). The concept is more than a decade old although it has been refined as modeling paradigms have evolved.

**4.2.8 COMPONENT DIAGRAM**

****

**EXPLANATION**

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems. User gives main query and it converted into sub queries and sends through data dissemination to data aggregators. Results are to be showed to user by data aggregators. All boxes are components and arrow indicates dependencies.

**4.2.9 DATA FLOW DIAGRAM**

**Level 0**

XGBoost Model

(prediction)

Result And Graphs

( legit / Phish )

User

Phishing Detection System

**Level 1**

Results & Graphs

Uploading Data

Configuring the Algorithm

Computational Speed

Fig 4.9: Data Flow Diagrams

**EXPLANATION:**

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.

**4.2.10 DEPLOYMENT DIAGRAM**



**EXPLANATION:**

Deployment Diagram is a type of diagram that specifies the physical hardware on which the software system will execute. It also determines how the software is deployed on the underlying hardware. It maps software pieces of a system to the device that are going to execute it.

**SYSTEM ARCHITECTURE:**

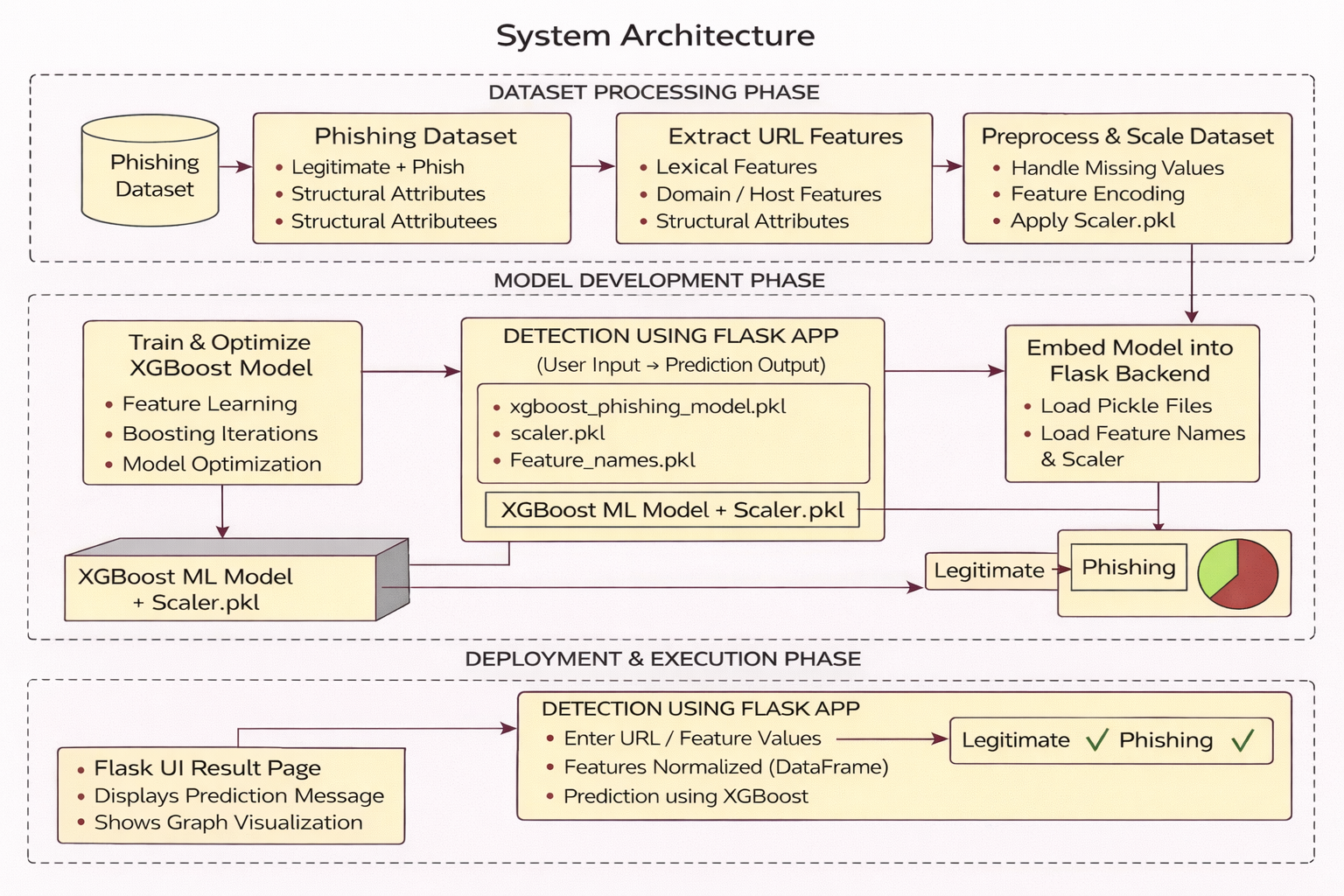


Fig 4.11: System Architecture

**CHAPTER 5**

**DEVELOPMENT TOOLS**

**5.1 Python**

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.

## 5.2 History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

#### 5.3 Importance of Python

* **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
* **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
* **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

#### 5.4 Features of Python

* **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
* **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
* **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
* **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
* **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
* **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
* **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
* **Databases** − Python provides interfaces to all major commercial databases.
* **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
* **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

* It supports functional and structured programming methods as well as OOP.
* It can be used as a scripting language or can be compiled to byte-code for building large applications.
* It provides very high-level dynamic data types and supports dynamic type checking.
* IT supports automatic garbage collection.
* It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

**5.5 Libraries used in python**

* numpy - mainly useful for its N-dimensional array objects.
* pandas - Python data analysis library, including structures such as dataframes.
* matplotlib - 2D plotting library producing publication quality figures.
* scikit-learn - the machine learning algorithms used for data analysis and data mining tasks.



Figure : NumPy, Pandas, Matplotlib, Scikit-learn

**CHAPTER 6**

**IMPLEMENTATION**

**6.1 GENERAL**

**Coding:**

**CHAPTER 7**

**SNAPSHOTS**

**General:**

This project is implements like application using python and the Server process is maintained using the SOCKET & SERVERSOCKET and the Design part is played by Cascading Style Sheet.

**SNAPSHOTS**

**CHAPTER 8**

**SOFTWARE TESTING**

**8.1 GENERAL**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**8.2 DEVELOPING METHODOLOGIES**

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

**8.3Types of Tests**

**8.3.1 Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**8.3.2 Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

**8.3.3 System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**8.3.4 Performance Test**

The Performance test ensures that the output be produced within the time limits,and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

**8.3.5 Integration Testing**

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**8.3.6 Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Acceptance testing for Data Synchronization:**

* The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
* The Route add operation is done only when there is a Route request in need
* The Status of Nodes information is done automatically in the Cache Updation process

**8.2.7 Build the test plan**

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identity the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from errors.

**CHAPTER 9**

**FUTURE ENHANCEMENT**

**9.1 FUTURE ENHANCEMENTS:**

In the future, this phishing detection system can be enhanced by integrating additional behavioral and content-based features such as webpage content analysis, DNS activity patterns, and user interaction signals to improve detection accuracy. The model can be extended to support real-time streaming data, allowing continuous monitoring of newly generated phishing URLs. Future work may also include the development of an interactive browser extension or mobile security module for on-device phishing alerts. Cross-dataset validation and ensemble learning techniques can be applied to further strengthen robustness against evolving phishing strategies. The system can be enhanced with explainable AI techniques to provide clearer justification for classification results. Cloud-based deployment and API integration may enable large-scale organizational security implementation. Additionally, periodic retraining mechanisms can be incorporated to adapt to emerging phishing trends. Overall, these enhancements will support improved adaptability, scalability, and real-world applicability of the phishing detection framework.

**CHAPTER 10**

**CONCLUSIONAND REFERENCES**

**10.1 CONCLUSION**

This project presents an effective and interpretable phishing URL detection system developed using a feature-based machine learning approach with the XGBoost classifier. By analyzing lexical, structural, and domain-level attributes of URLs, the model successfully distinguishes phishing websites from legitimate ones with high accuracy and reduced computational cost. The preprocessing, feature handling, and hyperparameter tuning stages contributed to improved model performance, ensuring reliable classification outcomes suitable for real-time use cases. Compared to deep learning-based approaches, the proposed system offers greater transparency by identifying key features that influence prediction results, thereby supporting informed and evidence-driven cybersecurity analysis. The evaluation using accuracy, precision, recall, F1-score, and confusion matrix metrics demonstrated the robustness and consistency of the model across the dataset. The lightweight nature of the framework enhances its feasibility for deployment in browser extensions, email filters, and web-security platforms. This work highlights the effectiveness of feature-optimized boosting models for phishing detection in tabular environments and reinforces the importance of data preprocessing and feature engineering in cyber threat classification. Overall, the project provides a practical, scalable, and deployment-ready phishing detection solution that contributes to improving online safety and reducing phishing-related risks. The outcomes of this study further encourage the adoption of machine learning-based approaches in cybersecurity applications and establish a foundation for future enhancements such as real-time monitoring, cross-dataset validation, and integration with advanced threat-intelligence systems.

**10.2 REFERENCES**

[1] R.D.Corin.Architecture of the WideandDeepNeuralNetwork.Accessed: May 5, 2024. [Online]. Available: https://www.researchgate.net/figure/ Architecture-of-the-wide-and-deep-neural-network-The-prediction-of the-sensors-states-is\_fig1\_337945667

[2] Creative Commons Attribution 4.0 Int. TabNet Architecture Consisting of the Encoder for Classification. Accessed: May 5, 2024. [Online]. Avail able: https://www.researchgate.net/figure/TabNet-architecture-consisting of-the-encoder-for-classification-This-is-composed-of\_fig1\_371301234

[3] Creative Commons Attribution-NonCommercial 4.0 Int. Deep Neural Network (DNN) Model Architecture. Accessed: May 5, 2024. [Online]. Available: https://www.researchgate.net/figure/Deep-Neural-Network DNN-model-architecture\_fig1\_370641687

[4] learnopencv. An Example of FNN With One Hidden Layer. Accessed: May5,2024. [Online]. Available: https://learnopencv.com/understanding feedforward-neural-networks/

[5] A. Yadav. (Apr. 22, 2023). What is Web Application. Accessed: May 5, 2024. [Online]. Available: https://medium.com/@akashyadav 1452/what-is-web-application-5f912f18f50f

[6] H.-J. Ye, S.-Y. Liu, H.-R. Cai, Q.-L. Zhou, and D.-C. Zhan, ‘‘A closer look at deep learning methods on tabular datasets,’’ 2024, arXiv:2407.00956.

[7] S.DangwalandA.-N.Moldovan,‘‘Featureselection for machinelearning based phishing websites detection,’’ in Proc. Int. Conf. Cyber Situational Awareness, Data Anal. Assessment (CyberSA), Dublin, Ireland, Jun. 2021, pp. 1–6.

[8] Y. Wei and Y. Sekiya, ‘‘Feature selection approach for phishing detection based on machine learning,’’ in Proc. Int. Conf. Appl. CyberSecur. (ACS), H.R.HassenandH.Batatia,Eds.,Cham,Switzerland:Springer,Jan.2022, pp. 61–70.

[8] M. R. Chinguwo and R. Dhanalakshmi, ‘‘Detecting cloud based phishing attacks using stacking ensemble machine learning technique,’’ Int. J. Res. Appl. Sci. Eng. Technol., vol. 11, no. 3, pp. 360–367, Mar. 2023, doi: 10.22214/ijraset.2023.49422.

[9] C. Rajeswary and M. Thirumaran, ‘‘The LSTM-based automated phishing detection driven model for detecting multiple attacks on tor hidden services,’’ J. Intell. Fuzzy Syst., vol. 44, no. 6, pp. 8889–8903, Mar. 2023, doi: 10.3233/jifs-224142.

[10] B. Subba, ‘‘A heterogeneous stacking ensemble-based security framework for detecting phishing attacks,’’ in Proc. Nat. Conf. Commun. (NCC), Feb. 2023, pp. 1–6, doi: 10.1109/NCC56989.2023.10068026.

[11] K. R. Nataraj, D. K. Yashaswini, R. Hema, N. S. Pawar, and S. Yashaswi, ‘‘Phishing attack detection using machine learning,’’ in Proc. 4th Int. Conf. Data Sci., Mach. Learn. Appl. Singapore: Springer, 2023, pp. 355–370, doi: 10.1007/978-981-99-2058-7\_33.

[12] D. T. Mosa, M. Y. Shams, A. A. Abohany, E.-S. M. El-Kenawy, and M. Thabet, ‘‘Machine learning techniques for detecting phishing URL attacks,’’ Comput., Mater. Continua, vol. 75, no. 1, pp. 1271–1290, 2023, doi: 10.32604/cmc.2023.036422.

[13] S. Yu, C. An, T. Yu, Z. Zhao, T. Li, and J. Wang, ‘‘Phishing detection based on multi-feature neural network,’’ in Proc. IEEE Int. Perform., Comput., Commun. Conf. (IPCCC), Nov. 2022, pp. 73–79, doi: 10.1109/IPCCC55026.2022.9894337.

[14] J. Novakovic and S. Markovic, ‘‘Detection of URL-based phishing attacks using neural networks,’’ in Proc. Int. Conf. Theor. Appl. Comput. Sci. Eng. (ICTASCE), Sep. 2022, pp. 132–136.

[15] S. A. Salihu, I. D. Oladipo, A. A. Wojuade, M. AbdulRaheem, A. O. Babatunde, A. R. Ajiboye, and G. B. Balogun, ‘‘Detec tion of phishing URLs using heuristics-based approach,’’ in Proc. 5th Inf. Technol. Educ. Develop. (ITED), Nov. 2022, pp. 1–7, doi: 10.1109/ITED56637.2022.10051199.

[16] J. Tanimu and S. Shiaeles, ‘‘Phishing detection using machine learning algorithm,’’ in Proc. IEEE Int. Conf. Cyber Secur. Resilience (CSR), Jul. 2022, pp. 317–322, doi: 10.1109/CSR54599.2022.9850316.

[17] M. Sánchez-Paniagua, E. Fidalgo, E. Alegre, and R. Alaiz-Rodríguez, ‘‘Phishing websites detection using a novel multipurpose dataset and web technologies features,’’ Expert Syst. Appl., vol. 207, Nov. 2022, Art. no. 118010, doi: 10.1016/j.eswa.2022.118010.

[18] M. Almseidin, A. A. Zuraiq, M. Al-Kasassbeh, and N. Alnidami, ‘‘Phishing detection based on machine learning and feature selection methods,’’ Int. J. Interact. Mobile Technol. (iJIM), vol. 13, no. 12, p. 171, Dec. 2019. [Online]. Available: https://www.learntechlib.org/p/216410

[19] R. Jayaraj, A. Pushpalatha, K. Sangeetha, T. Kamaleshwar, S. U. Shree, and D. Damodaran, ‘‘Intrusion detection based on phishing detection with machine learning,’’ Meas., Sensors, vol. 31, Feb. 2024, Art. no. 101003. [Online]. Available: https://www.sciencedirect.com/ science/article/pii/S2665917423003392

[20] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, ‘‘Deep neural networks and tabular data: A survey,’’ 2021, arXiv:2110.01889.

[21] F.Salahdine,Z.E.Mrabet,andN.Kaabouch,‘‘Phishingattacksdetectiona machine learning-based approach,’’ in Proc. IEEE 12th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON), New York, NY, USA, Dec. 2021, pp. 250–255.

[22] M. Baykara and Z. Z. Gürel, ‘‘Detection of phishing attacks,’’ in Proc. 6th Int. Symp. Digit. Forensic Secur. (ISDFS), Antalya, Turkey, Mar. 2018, pp. 1–5.

[23] K. L. Chiew, K. S. C. Yong, and C. L. Tan, ‘‘A survey of phishing attacks: Their types, vectors and technical approaches,’’ Expert Syst. Appl., vol. 106, pp. 1–20, Sep. 2018.

[24] F. Yahya, ‘‘Detection of phishing websites using machine learning approaches,’’ in Proc. Int. Conf. Data Sci. Appl. (ICoDSA), Bandung, Indonesia, 2021, pp. 40–47.

[25] A. Alswailem, B. Alabdullah, N. Alrumayh, and A. Alsedrani, ‘‘Detecting phishing websites using machine learning,’’ in Proc. 2nd Int. Conf. Comput. Appl. Inf. Secur. (ICCAIS), Riyadh, Saudi Arabia, May 2019, pp. 1–6.