A Machine Learning Framework for Web URL Security Using Feature Engineering

Project Report

Submitted to the APJ Abdul Kalam Technological University

in partial fulfillment of requirements for the award of degree

Bachelor of Technology

in

Computer Science and Engineering

by

ABBY ANISH VARGHESE (SCT20CS001)

ANANTHAN UNNI M (SCT20CS012)

ARUL R (SCT20CS016)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SREE CHITRA THIRUNAL COLLEGE OF ENGINEERING

PAPPANAMCODE, THIRUVANANTHAPURAM, KERALA

MAY 2024

DECLARATION

We, the undersigned, affirmatively state that the project report titled "A Machine Learning Framework for Web URL Security Using Feature Engineering", submitted in partial fulfillment of the requirements for the Bachelor of Technology degree at APJ Abdul Kalam Technological University, Kerala, is a genuine and original work completed by us under the guidance of Mrs. Rejimol Robinson R R. The content of this submission is an expression of our own thoughts, and in cases where external ideas or verbiage are incorporated, proper citation and referencing of the original sources have been diligently and accurately performed. We further assert that we have strictly adhered to the principles of academic honesty and integrity throughout the research process. No data, idea, fact, or source has been misrepresented or fabricated in our submission. We acknowledge that any breach of the aforementioned principles may result in disciplinary measures by the institute and/or the University, and may also lead to penal actions from sources that have not been appropriately cited or from which proper permissions have not been obtained. It is confirmed that this report has not been utilized previously as the basis for obtaining any degree, diploma, or similar title from any other University.

Place: Thiruvananthapuram

Date: 6/5/2024

Abby Anish Varghese

Ananthan Unni M

Arul R

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

SREE CHITRA THIRUNAL COLLEGE OF ENGINEERING

THIRUVANANTHAPURAM

2023 - 24

CERTIFICATE

This is to certify that the report entitled **A Machine Learning Framework for Web URL Security Using Feature Engineering** submitted by **Abby Anish Varghese (SCT20CS001)**, **Ananthan Unni M (SCT20CS012)**, and **Arul R (SCT20CS016)** to the APJ Abdul Kalam Technological University in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering is a bonafide record of the project work carried out by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Prof. Rejimol Robinson.R.R
(Seminar Guide)
Associate Professor
Dept.of CSE
Sree Chitra Thirunal
College of Engineering
Thiruvananthapuram

Dr.Chitharanjan Karat
(Seminar Coordinator)
Assistant Professor
Dept.of CSE
Sree Chitra Thirunal
College of Engineering
Thiruvananthapuram

ACKNOWLEDGEMENT

We take this opportunity to express our deepest sense of gratitude and sincere thanks to everyone who helped us to complete this work successfully. We express our sincere thanks to **Dr. Soniya B**, Head of Department, Computer Science and Engineering, Sree Chitra Thirunal College of Engineering for providing us with all the necessary facilities and support.

We would like to place on record our sincere gratitude to our project guide **Prof.Rejimol Robinson R R**, Associate Professor, Computer Science and Engineering, Sree Chitra Thirunal College of Engineering for her guidance and mentorship throughout this work.

Finally we thank our family, and friends who contributed to the succesful fulfilment of this seminar work.

Abby Anish Varghese

Ananthan Unni M

Arul R

ABSTRACT

In the ever-evolving landscape of cybersecurity, effective URL classification and threat detection remain critical to safeguarding digital infrastructures against malicious attacks. This project represents a comprehensive endeavor to address these challenges by integrating advanced machine learning techniques, explainable AI methodologies, and interactive user interfaces. Central to our methodology is a meticulously curated dataset comprising URLs categorized into five distinct classes: benign, malware, spam, phishing, and defacement. Leveraging this dataset, we conduct feature extraction, identifying 34 informative features crucial for classification. These features are then organized into a structured CSV file, laying the groundwork for subsequent analysis. We employ a diverse array of machine learning algorithms, including Random Forest, XGBoost, CatBoost, LSTM, and CNN-LSTM, to analyze the dataset and derive insights. Notably, Random Forest emerges as the most effective model, showcasing superior performance in URL classification tasks. To enhance transparency and interpretability, we incorporate Explainable AI techniques, leveraging SHAP (SHapley Additive exPlanations) graphs to visualize feature importance. This facilitates a deeper understanding of the underlying decision-making processes of the Random Forest model, instilling confidence in its outcomes. Building upon these insights, we seamlessly integrate Google's Gemini model into our framework. This interactive model provides users with a user-friendly interface capable of addressing queries in real-time, enhancing the accessibility and usability of our solution. The SHAP graphs derived from the Random Forest model serve as input for the Gemini model, enriching its predictive capabilities and facilitating more informed decision-making. Furthermore, we design our system as a plugin model, allowing users to input URLs directly for classification. Additionally, an intuitive text-based query system, powered by Google's Language Model, provides users with dynamic responses and insights into the classification process, further enhancing user engagement and understanding. In conclusion, our project represents a significant advancement in cybersecurity methodologies, offering a comprehensive solution for URL classification and threat detection. By combining state-of-the-art machine learning techniques with explainable AI and interactive interfaces, we empower users with enhanced security measures and actionable insights, thereby fortifying digital defenses in an increasingly hostile cyber landscape.

Contents

List of Figures

Chapter 1

INTRODUCTION

The rapid proliferation of cyber threats poses significant challenges to the integrity and security of digital ecosystems worldwide. Among these threats, the classification of URLs into distinct categories such as benign, malware, spam, phishing, and defacement is paramount for effective threat mitigation and risk management. In response to this imperative, our project endeavors to develop an advanced framework for URL classification and threat detection through a synergistic integration of machine learning algorithms, explainable AI techniques, and interactive interfaces. Central to our approach is a meticulously curated dataset comprising URLs categorized into the aforementioned classes. Through rigorous feature extraction, we distill 34 informative features from these URLs, capturing diverse attributes essential for comprehensive classification. The culmination of this process yields a single labeled CSV file, meticulously sorted into columns, laying the foundation for subsequent analysis. Our exploration spans a spectrum of machine learning algorithms, including Random Forest, XGBoost, CatBoost, LSTM, and CNN-LSTM. Through systematic experimentation and evaluation, Random Forest emerges as the optimal model, demonstrating superior performance in URL classification tasks. Leveraging the output of the Random Forest model, we incorporate Explainable AI techniques, particularly SHAP (SHapley Additive exPlanations) graphs, to elucidate feature importance and enhance interpretability. Building upon these insights, we integrate Google's Gemini model into our framework, augmenting its predictive capabilities and enabling an interactive interface for user interaction. The Gemini model, renowned for its robustness and versatility, offers users the ability to pose queries in real-time, enhancing the accessibility and usability of our solution. Moreover, the incorporation of a plugin model design streamlines integration into existing cybersecurity architectures, providing users with a seamless and intuitive platform for URL classification and threat detection. In summary, by leveraging the synergies between machine learning, explainable AI, and interactive interfaces, we aim to empower users with the tools and insights necessary to navigate the complex landscape of online threats with confidence and efficacy.

Chapter 2

LITERATURE REVIEW

2.1 Using Lexical Features for Malicious URL Detection

This paper suggests employing a machine learning strategy to identify and obstruct malicious URLs. The method involves analyzing static lexical features derived from the URL string to effectively counter the prevalence of cyber-attacks and scams associated with such URLs. By assuming that the distribution of static lexical features differs between malicious and benign URLs,
Implemented in the FireEye Advanced URL Detection Engine (FAUDE), the model has demonstrated a significant improvement in detecting malicious URLs. The paper concludes that a Random Forest Classification approach, relying on static lexical features, proves effective in distinguishing between malicious and benign URLs. The outcomes highlight the potential of purely lexical methods in lightweight systems, enabling rapid real-time assessments of URLs or websites on the Internet.
Notably, the Random Forest model surpasses other machine learning classifiers in terms of accuracy and false negative rate. The integration of 1000 trigram-based features with lexical features successfully meets the desired criteria of a low false negative rate, with a reasonable compromise on the false positive rate and achieving high accuracy.

2.2 Research on Malicious URL Detection Based on Feature Contribution Tendency

The paper focuses on the detection of malicious URLs through feature selection and the application of machine learning classification models. Initially, features that distinguish malicious URLs from benign ones are identified. Three types of URL features are extracted, including structure features (e.g., pure domain, top-level domain), intelligence features (e.g., Alexa ranking, Whois information), and sensitive word features. A total of 28 URL features are extracted for analysis.

Seven machine learning classification models are selected for comparative experiments on the dataset, and evaluation indexes are calculated to determine the model with the best performance. Random Forest is identified as having better performance in malicious URL detection, leading to its selection for further research.

Hyperparameter optimization is conducted using Bayesian search, utilizing the Bayesian Optimization module in the sklearn library. The objective function is set to find the optimal combination of model parameters, with a focus on reducing calculation time by constructing possible assignment ranges for key parameters.

Data collection is conducted through various channels. Benign samples are collected by visiting the top 200,000 websites using Alexa, while 200,000 malicious URLs are gathered as the malicious sample set. This comprehensive approach to data collection ensures the diversity and Representativeness of the dataset for training and testing the classification models.

In conclusion, the paper presents a systematic approach to malicious URL detection, incorporating feature selection, machine learning model selection, and Hyperparameter optimization. By leveraging a diverse range of URL features and employing advanced techniques such as Bayesian search, the proposed method demonstrates effectiveness in distinguishing between malicious and benign URLs, contributing to the advancement of Cybersecurity measures.

2.3 Spam and Non-Spam URL Detection using Machine Learning Approach

The project aims to enhance understanding of phishing attacks and improve detection methods through the development of a tool utilizing machine learning and deep neural networks. This tool classifies Uniform Resource Locator (URLs) as either phishing or legitimate, leveraging datasets collected from various sources containing examples of both types of URLs. These datasets are then used to train supervised learning models, and the accuracy of various classification algorithms is evaluated.

The classification algorithms utilized in the project include decision trees, random forest trees, Multilayer Perceptron (MLP), XGBoost, and Support Vector Machine (SVM). The effectiveness of these algorithms is assessed primarily based on accuracy metrics, with XGBoost demonstrating the highest accuracy among the models tested. Specifically, the XGBoost model achieves an accuracy of 86.5% on the training dataset. This suggests that XGBoost performs well when dealing with datasets containing both categorical and numeric values, or solely numeric values.

Phishing attacks represent a significant and escalating threat due to widespread unawareness among individuals, leading them to inadvertently open phishing emails and visit fraudulent websites. As phishing techniques evolve alongside technological advancements, detection becomes increasingly challenging. However, awareness-raising efforts are crucial to combatting this threat. The proposed project addresses this challenge by training a neural network to classify URLs as either phishing or legitimate, with a focus on identifying the most accurate model for this purpose. The models developed in the project achieve accuracies ranging from 80% to 86%, with XGBoost emerging as the top-performing algorithm. XGBoost, a gradient boosting algorithm, is highlighted as particularly dominant in contemporary applied machine learning.

The project contributes to the ongoing effort to combat phishing attacks by developing a tool capable of accurately classifying URLs. By leveraging machine learning and deep neural networks, the project underscores the importance of awareness and effective detection methods in mitigating the growing threat of phishing attacks in modern society.

2.4 Machine Learning Algorithms Evaluation for Phishing URLs Classification

The paper evaluates eleven machine learning algorithms commonly used for detecting and classifying phishing URLs. These algorithms include Decision Tree (DT), Nearest Neighbors (KNN), Gradient Boosting (GB), Logistic Regression (LR), Naïve Bayes (NB), Random Forest (RF), Support Vector Machines (SVM), Neural Network (NN), Extra Tree (ET), Ada Boost (AB), and Bagging (B). The evaluation is performed using Python as the implementation tool, and phishing URLs and benign URLs are collected from sources such as PhishTank.com and OpenPhish.org. Lexical analysis is employed to extract URL features, and data preprocessing techniques, including scaling, are utilized to prepare the data for input.

After data preparation, the selected machine learning algorithms are implemented and trained on the prepared dataset. Cross-validation techniques are used to evaluate the performance of the models. The paper focuses on accuracy as the performance metric, comparing the performance of each algorithm based on accuracy scores. The results are presented in a table format to facilitate comparison.

Among the evaluated algorithms, Extra Tree and K-Nearest Neighbors (KNN) achieve the highest accuracy score of 91%. This indicates that these algorithms demonstrate superior performance in accurately classifying phishing and benign URLs compared to the other algorithms evaluated. The paper highlights the effectiveness of these algorithms in phishing URL classification tasks and underscores the importance of selecting appropriate machine learning algorithms for achieving high accuracy in detecting phishing attacks.

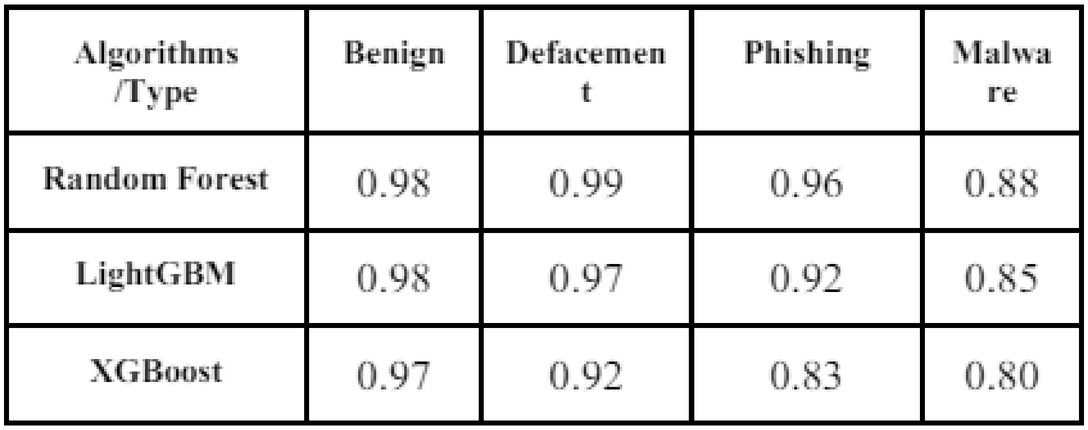
Overall, the paper provides valuable insights into the performance of various machine learning algorithms in detecting and classifying phishing URLs, offering guidance to researchers and practitioners in selecting suitable algorithms for their detection systems.

2.5 Malicious URL Detection and Classification Analysis using Machine Learning Models

The proliferation of malicious websites and URLs poses a significant cybersecurity threat, leading to substantial financial losses and potential scams for unsuspecting users. The need for a robust system capable of categorizing and identifying these dangerous URLs is paramount as phishing, spamming, and malware attacks continue to rise. However, the complexity of these threats, including the vast amount of data, evolving patterns, non-linear relationships, and the absence of comprehensive training data, presents a significant challenge in effective classification. In a recent study, malicious URLs were categorized across different types such as Phishing, Benign, Defacement, and Malware, using a dataset comprising over 651,191 URLs. Machine learning algorithms like random forest, LightGBM, and XGBoost were employed to detect and classify these malicious URLs, aiming to enhance cybersecurity measures.

The evaluation of different machine learning classifiers revealed varying performance metrics across different types of malicious URLs. The random forest classifier exhibited superior precision, particularly benefiting from balanced training data that includes a nearly equal number of harmful and benign websites. On the other hand, the LightGBM classifier leveraged a leaf-wise distribution of the dataset, showcasing high precision in classifying Benign URLs. Moreover, these classifiers demonstrated robust recall rates across different malicious URL types, with LightGBM excelling in Benign and Defacement types, and Random Forest performing well in Defacement, Phishing, and Malware categories. The F1 score, a combined metric of precision and recall, further emphasized the effectiveness of random forest and LightGBM classifiers in accurately classifying various types of URLs, especially those with different malicious intents.

The findings underscore the importance of employing machine learning techniques, such as random forest and LightGBM, for efficient detection and classification of malicious URLs. These models can be integrated into search engines or website security systems to proactively identify and alert users about potential fraudulent URLs, thereby enhancing cybersecurity measures and safeguarding users from online threats. Future research directions may focus on refining feature selection techniques, exploring ensemble methods, and optimizing model performance to address evolving cybersecurity challenges effectively.



Algorithms /Type	Benign	Defacement	Phishing	Malware
Random Forest	0.98	0.99	0.96	0.88
LightGBM	0.98	0.97	0.92	0.85
XGBoost	0.97	0.92	0.83	0.80

Figure 2.1: Analysis of F1 Score

2.6 A Hybrid approach combining blocklists, machine learning and deep learning for detection of malicious URLs

The paper presents a hybrid approach for detecting malicious URLs, which combines traditional blocklist-based methods with machine learning (ML) and deep learning (DL) techniques. The traditional approach involves using a blocklist of known malicious URLs, while the ML and DL methods are employed to detect new malicious URLs for which no signatures exist. The system is trained and tested on a large dataset comprising 2.5 million malicious and benign URLs, achieving precision, recall, and F1-score greater than 0.97.

Malicious and benign URLs are collected from various online sources, including Phistank, GitHub repositories, and the ISCX-URL20161 dataset. The collected URLs undergo deduplication to remove duplicates and are then divided into positive (malicious) and negative (benign) samples. The positive set is used for the blocklist module, where URLs are hashed and stored in an in-memory database. When a URL is encountered, it is checked against the blocklist; if found, it is flagged as malicious. If not, it is passed to the ML module for feature extraction and classification.

The ML module extracts features from the URL and performs classification to determine whether the URL is malicious or benign. The hybrid approach offers the benefits of both traditional blocklist methods and modern ML techniques. Shallow ML models achieve an accuracy of 0.97, while deep ML models achieve an accuracy of 0.98. The results suggest that with proper feature extraction, shallow ML algorithms can achieve accuracy comparable to deep learning algorithms for detecting malicious URLs.

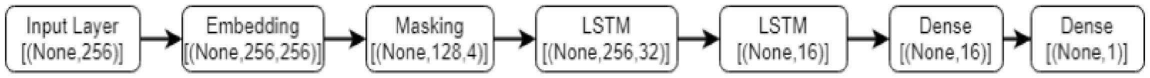


Figure 2.2: LSTM network architecture

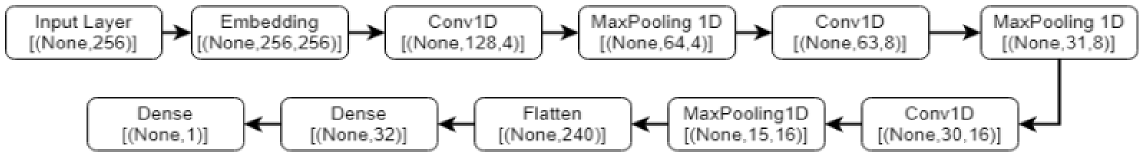


Figure 2.3: CNN Network Architecture

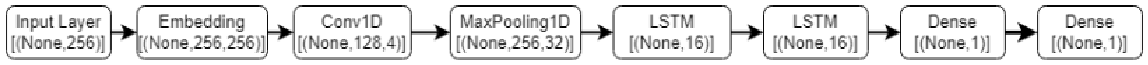


Figure 2.4: CNN-LSTM Network

Future plans include extending the approach to incorporate more state-of-the-art deep learning algorithms and improving feature extraction for shallow ML models. Additionally, the development of a plugin-based system deployable as a browser extension or reverse proxy-based system is proposed. Overall, the hybrid approach presented in the paper demonstrates promising results in detecting malicious URLs and suggests avenues for further research and development in this domain

2.7 Phishing URL recognition based on ON-LSTM attention mechanism and XGBoost model

[**p4**]The paper presents a method for identifying phishing URLs through a multi-step approach. Firstly, URLs are segmented at the word level, and hierarchical features are extracted using an unsupervised self-learning process with the ON-LSTM model. This extraction process aims to fully capture the characteristics of URLs. Subsequently, an attention mechanism is applied to assign weights to the feature vectors obtained from the ON-LSTM model, enhancing the depth of URL feature extraction by focusing on important features. Finally, a classification model based on XGBoost is constructed using the feature vectors to classify and recognize URLs efficiently.

Data Pre-processing involves segmenting URLs based on sensitive words, followed by normalization of the word set by data length. Word2Vec word embedding and dimension reduction are then applied to obtain a matrix of word vectors representing URLs. Feature extraction is conducted using the ON-LSTM algorithm combined with an attention mechanism to extract hierarchical features from URLs. The attention mechanism ensures that feature extraction considers the importance of features, leading to the generation of a comprehensive feature vector for each URL.

For classification and recognition, the XGBoost algorithm is chosen due to its superior performance in classification tasks. The 65-dimensional word-level feature vector, along with ON-LSTM parameters such as 160 hidden layer nodes, 35 iterations, and a data batch size of 100, are used to train the XGBoost model. Additionally, dropout regularization with a rate of 0.3 is applied to prevent overfitting.

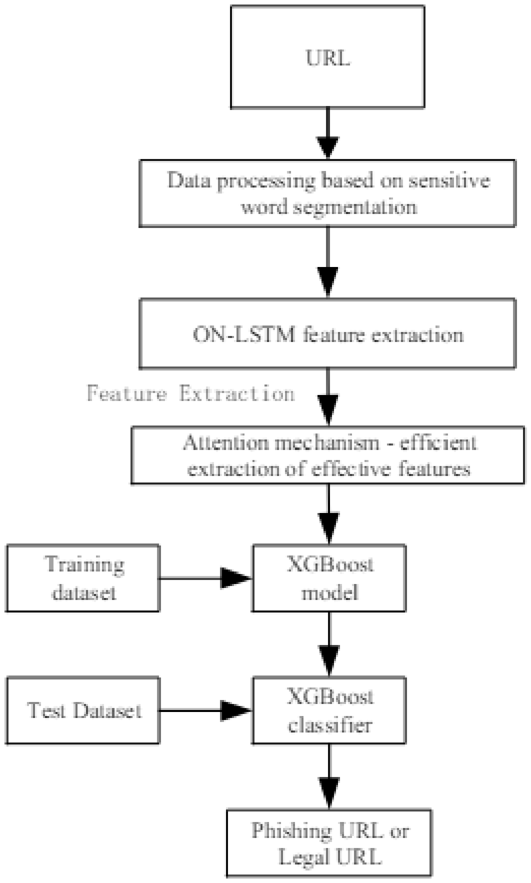

URL
Data processing based on sensitive word segmentation
ON-LSTM feature extraction
Feature Extraction
Attention mechanism - efficient extraction of effective features
Training dataset
XGBoost model
Test Dataset
XGBoost classifier
Phishing URL or Legal URL

Figure 2.5: A Phishing URL Recognition Model Based on ON-LSTM Attention Mechanism and XGBoost Model

Experimental evaluation is conducted to compare the proposed model with other baseline models, including LSTM, BiLSTM, BiLSTM with attention (BiLSTM att), and CNN BiLSTM with attention (CNN BiLSTM att). Results show that the proposed method outperforms these baseline models in terms of accuracy, precision, and recall. Particularly, the accuracy rate is significantly improved compared to the CNN BiLSTM att model, indicating the effectiveness of the ON-LSTM approach in capturing URL features comprehensively and enhancing phishing URL identification performance.

2.8 Phishing URL detection by leveraging RoBERTa for feature extraction and LSTM for classification

The study introduces an innovative approach to phishing URL detection by integrating advanced machine learning techniques. Specifically, it combines the RoBERTa transformer-based model for feature extraction with the LSTM architecture for classification. The RoBERTa model is adept at extracting semantic and contextual information from URLs, enabling it to capture intricate meanings and contextual nuances effectively. This feature extraction process generates contextualized embeddings that encode the semantic understanding and surrounding context of URLs. Subsequently, the LSTM layer is utilized for precise categorization by leveraging the sequential relationships captured from the extracted features.

The proposed system is trained and tested on an extensive dataset comprising 300,000 URLs, achieving an impressive accuracy rate of 97.14% in distinguishing between legitimate and phishing URLs. These results highlight the efficacy of combining LSTM for classification and RoBERTa for feature extraction in phishing URL detection, significantly advancing phishing detection techniques and offering practical countermeasures against online threats.

The methodology and experimental setup of the study are designed to demonstrate the superiority of the proposed approach over existing techniques. Comparative analysis against rule-based and traditional machine learning approaches showcases the robustness and reliability of the hybrid RoBERTa-LSTM model, significantly outperforming baseline methods. The high accuracy rate not only underscores its effectiveness in detecting phishing URLs but also emphasizes its potential as a powerful tool for organizations and individuals to enhance security measures and mitigate risks associated with phishing attacks.

Future research directions may include assessing the model's generalizability on diverse datasets and exploring advanced techniques like transfer learning and ensemble methods to further boost performance and scalability. These endeavors can contribute to building a safer digital environment and strengthening defenses against evolving cyber threats.

In conclusion, the successful integration of RoBERTa for feature extraction and LSTM for classification demonstrates a promising approach in phishing URL detection, offering practical implications for cybersecurity enhancement and risk mitigation strategies. The study's contributions extend to advancing phishing detection techniques, improving security protocols, and providing valuable insights for future research and development in the cybersecurity domain.

2.9 A Bi-Directional LSTM Model with Attention for Malicious URL Detection

The paper proposes a novel model, AB-BiLSTM with attention mechanism, for malicious URL detection. The model employs a multi-step approach involving preprocessing of URLs into word vectors using pre-trained Word2Vec, followed by training a Bidirectional Long Short-Term Memory (BiLSTM) network combined with an attention mechanism to extract features and classify the URL sequences. The proposed model was tested on a dataset consisting of both benign and malicious URLs, achieving high accuracy, precision, recall rates, and F1 score, outperforming traditional machine learning and deep learning models. The architecture of the AB-BiLSTM model comprises five components: input layer, embedding layer, BiLSTM layer, attention layer, and output layer. Dropout strategy is incorporated before the output layer to mitigate overfitting.

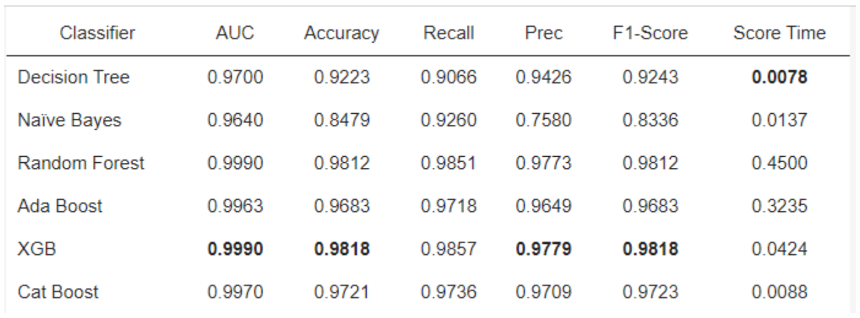
The labeled dataset used in the study consists of a total of 453,340 URLs, with 80% used for training and 20% for testing. Benign URLs were sourced from DMOZ, a volunteer-edited directory of the Web, while malicious URLs were obtained from VirusTotal and PhishTank. The paper compares the proposed AB-BiLSTM model with traditional feature-engineered models with logistic regression (LR), demonstrating superior performance. The AB-BiLSTM model consistently achieved higher accuracy, precision, recall rates, and F1 score, showcasing its effectiveness in malicious URL detection.

Furthermore, the paper compares the AB-BiLSTM model with other variations, including a Char-AB-BiLSTM model, which incorporates character-level embedding instead of word-level embedding. While both models show promising results, the AB-BiLSTM model outperforms the Char-AB-BiLSTM model, indicating the efficacy of word-level embedding, particularly with pre-trained Word2Vec embeddings. The study suggests that word embedding methods effectively capture contextual information in URLs, contributing to improved performance in malicious URL detection tasks.

The incorporation of an attention mechanism in the AB-BiLSTM model enables capturing long-distance dependencies within URL sequences and assigning higher weights to important features, enhancing its ability to discriminate between benign and malicious URLs. Overall, the paper highlights the advantages of neural network models, particularly those integrating attention mechanisms and word-level embeddings, in effectively detecting malicious URLs compared to traditional feature-engineered approaches.

2.10 Interpretable Machine Learning Models for Malicious Domains Detection Using Explainable Artificial Intelligence (XAI)

This paper delves into the realm of machine learning (ML) models for the detection of malicious domains, employing a comprehensive analysis of Decision Trees (DT), Naive Bayes (NB), Random Forest (RF), AdaBoost (AB), XGBoost (XGB), and CatBoost (CB). Our investigation is based on datasets comprising 45,000 samples each of malicious and non-malicious domains. To enhance model performance, feature selection was conducted using the forward feature selector. Two experiments were carried out, utilizing both the preprocessed data (27 features) and the reduced features data (10 features) after the feature selection process. The XGBoost algorithm, integrated with feature selection techniques, produced robust results—AUC of 0.999, accuracy of 0.9818, precision of 0.9779, and an F1-score of 0.9818, accompanied by a score time of 0.0424. The decision to choose XGBoost with feature selection was based on its optimal balance between performance and computational efficiency.



Classifier	AUC	Accuracy	Recall	Prec	F1-Score	Score Time
Decision Tree	0.9700	0.9223	0.9066	0.9426	0.9243	**0.0078**
Naïve Bayes	0.9640	0.8479	0.9260	0.7580	0.8336	0.0137
Random Forest	0.9990	0.9812	0.9851	0.9773	0.9812	0.4500
Ada Boost	0.9963	0.9683	0.9718	0.9649	0.9683	0.3235
XGB	**0.9990**	**0.9818**	0.9857	**0.9779**	**0.9818**	0.0424
Cat Boost	0.9970	0.9721	0.9736	0.9709	0.9723	0.0088

Figure 2.6: Classification results of the models after feature selection

In order to unravel the intricacies of our model, we employed XAI techniques for both global and local interpretation. The Global Surrogate Model, Shapley values, and Local Interpretable Model-agnostic Explanations (LIME) were employed to shed light on the decision-making process of the XGBoost model. XGBoost emerged as the top performer among all classifiers, achieving remarkable metrics with 27 features—AUC of 0.9991, accuracy of 0.9856, precision of 0.9836, and an F1 score of 0.9856. Notably, Random Forest achieved the highest recall of 0.9886 without feature selection. The significance of score time was underscored, and although XGBoost demonstrated comparable performance with and without feature selection, the latter exhibited significantly reduced score time. Consequently, XGBoost with feature selection was deemed the final model for explainable artificial intelligence (XAI).

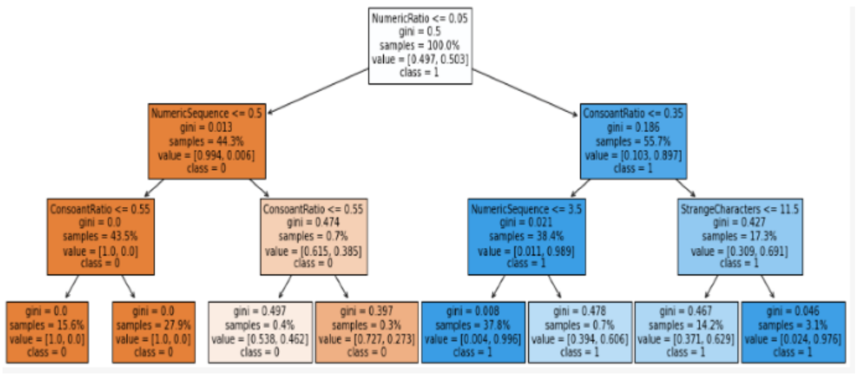


Figure 2.7: Global Surrogate Model using Decision Tree for XGB

2.11 Data Driven based Malicious URL Detection using Explainable AI

This paper presents a comprehensive approach to malicious URL detection, acknowledging the prevalent threat these URLs pose to system security. With a dataset comprising 651,191 URLs classified into benign, defacement, phishing, or malware categories, the study extracts 21 features from these URLs to train the proposed model. These features encompass various aspects such as URL length, count of special characters, and digits, reflecting the diversity of attack vectors encountered in malicious websites.

A significant contribution of the paper lies in the development of a two-stage stacked ensemble learning model, combining gradient boosting methods and random forest algorithms. Trained and tested on a 70:30 split of the dataset, the model achieves an impressive accuracy rate of 97%, underscoring its efficacy in distinguishing between benign and malicious URLs. Moreover, the use of Explainable AI (XAI) is introduced to elucidate the workings of the model and analyze the impact of each feature on the predictions across the four class categories.

The utilization of XAI represents a novel aspect of the proposed model, offering insights into the decision-making process of machine learning algorithms and demystifying their black-box nature. This approach enhances the interpretability and transparency of the model, facilitating a deeper understanding of its functioning and the factors influencing its predictions.

Given the persistent and escalating threat of malicious URLs, the research underscores the importance of continuous advancements in cyber-security measures. As cyberattacks continue to evolve alongside the increasing use of the internet, the study emphasizes the need for proactive research to develop robust defense mechanisms against such threats.

Future directions for research in this domain include enhancing the model's accuracy through the utilization of updated datasets and exploring real-time prediction capabilities using web crawling techniques. Additionally, the application of XAI in other research fields holds promise for achieving better results and improving interpretability across various use cases, underscoring its potential for broader impact beyond malicious URL detection.

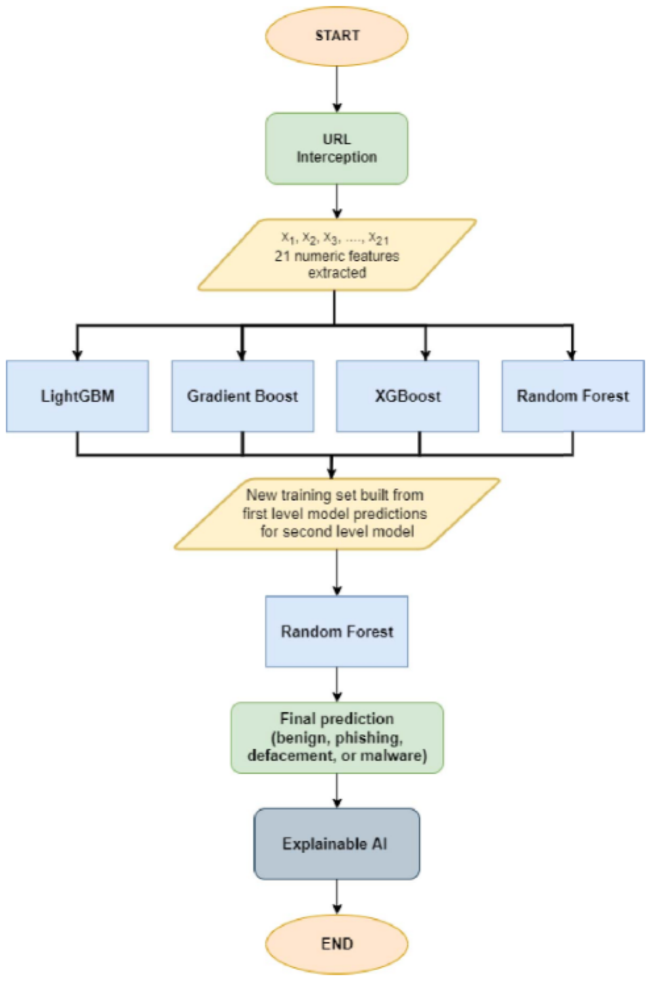

START
URL Interception
$x_1, x_2, x_3, \ldots, x_{21}$ 21 numeric features extracted
LightGBM
Gradient Boost
XGBoost
Random Forest
New training set built from first level model predictions for second level model
Random Forest
Final prediction (benign, phishing, defacement, or malware)
Explainable AI
END

Figure 2.8: Flowchart of Proposed Methadology

2.12 Phishing Detection Using URL-based XAI Techniques

This paper delves into the realm of phishing detection, employing explainable techniques such as Local Interpretable Model-Agnostic Explanations (LIME) and Explainable Boosting Machine (EBM) to shed light on recent advancements and future prospects in the field. The study utilizes the Ebbu2017 database, containing a collection of legitimate and phishing URLs, to train and test the proposed models. With a total of 73,575 URLs, pre-processing techniques are applied to prepare the data for analysis. Notably, the dataset encompasses 40 features ranging from keyword counts to domain name randomness, serving as input vectors for the machine learning models.

One key advantage of the system presented in this paper lies in its language independence and real-time execution capabilities. By leveraging features extracted via Natural Language Processing (NLP), the system can swiftly detect phishing attempts without relying on language-specific dictionaries or third-party software. Furthermore, the interpretability of the extracted features facilitates the development of transparent models, enabling stakeholders to comprehend the decision-making process.

To implement the LIME and EBM models, the study employs the InterpretML package, a comprehensive framework for interpretability in machine learning. Traditional algorithms such as Support Vector Machine (SVM) and Random Forest are evaluated within the LIME approach due to their popularity and effectiveness on the feature set. The data is partitioned into training and test sets in a 90-10 ratio to facilitate model evaluation.

The experiments conducted demonstrate the efficacy of the proposed models in accurately classifying URLs as phishing or legitimate while providing explainability to the decision-making process. By shedding light on the underlying mechanisms driving the models' predictions, the study enhances the interpretability and trustworthiness of machine learning models in phishing detection.

For future research directions, the study suggests exploring new models trained on the most relevant features to enhance speed and accuracy. Additionally, further investigation into AI techniques and machine learning models could lead to the development of more robust and efficient phishing detection systems, paving the way for improved cybersecurity measures.

Chapter 3

METHODOLOGY

In this chapter, we outline our approach to URL classification, which began by sourcing a dataset containing URLs categorized into five distinct classes: benign, malware, spam, phishing, and defacement.We performed feature extraction on this dataset, meticulously extracting 34 features that were relevant for classification purposes. These features were then organized into a single labeled CSV file, laying the groundwork for subsequent analysis. We collectively explored the application of various machine learning algorithms to our dataset. This included Random Forest, XGBoost, CatBoost, LSTM, and CNN-LSTM. Through experimentation and discussion, we evaluated the performance of each algorithm, considering factors such as accuracy and computational efficiency. Upon identifying the most effective models, we worked together to integrate Explainable AI techniques into our framework. Utilizing SHAP graphs, we extracted feature importance insights from our models' outputs, enhancing their interpretability and transparency. This allowed us to gain a deeper understanding of the underlying patterns driving our models' predictions. We integrated Google's Gemini model into our framework. With its interactive interface capable of addressing user queries, we saw an opportunity to enhance the user experience of our classification system. Later we designed a plugin model that allows users to input URLs for classification and interact with the system through a text-based interface.

In the subsequent sections, we shall delve into intricate terminologies and delineate the meticulous processes integral to our project.

3.1 ISCX-2016 Dataset

We began by sourcing the URLs from the ISCX-2016 dataset provided by the Canadian Institute of Cybersecurity. The dataset was created using the Heritrix web crawler to collect over 35,300 URLs from top Alexa websites, followed by a meticulous deduplication process and validation through VirusTotal. The WEBSPAM-UK2007 dataset served as the source for spam URLs, from which over 12,000 URLs were gathered. OpenPhish, a renowned repository of phishing sites, provided approximately 10,000 phishing URLs, ensuring the dataset's currency and relevance. Meanwhile, the DNS-BH project contributed over 11,500 malware-related URLs, maintaining a meticulous list of malware sites. Lastly, more than 45,450 defacement URLs were compiled from trusted Alexa-ranked websites, where fraudulent or obscured URLs hosted malicious web pages. Each dataset underwent careful curation and organization into labeled CSV files for accessibility and ease of analysis.

3.2 Feature Engineering

The feature extraction process involves analyzing URLs from various categories, namely **Defacement**, **Phishing**, **Malware**, **Spam**, and **Benign**, stored in separate CSV files. A `UrlFeaturizer` class is employed to extract a diverse range of features from each URL, encompassing both URL string features and domain-related features.

Firstly, the URL is dissected to derive its domain. Then, utilizing the `whois` module, information such as **creation_date** and **expiration_date** of the domain is obtained, if available. Subsequently, `requests` and `PyQuery` are utilized to fetch and parse the webpage content, facilitating the extraction of page-related features like **body_length**, **num_titles**, **num_images**, **num_links**, and **script_length**.

Additionally, various string-based features such as **url_entropy**, presence of **HTTP/HTTPS**, existence of **digits**, **parameters**, **fragments**, and **subdomains** are determined. Domain-specific features include **days_since_registration** and **days_since_expiration**, the presence and count of specific characters, **extensions**, the presence of keywords like **"server"** or **"client"**, and the number of **NameServers** and **MailServers** resolved.

Network-related features such as **SSL** certificate validity, **redirects**, **ASN** (**Autonomous System Number**) associated with the IP, **IP geolocation**, **PTR** (**Pointer Record**) associated with the IP, and presence on **RBL** (**Real-time Blackhole List**) are also examined. Furthermore, the URL and domain are checked for **blacklisting** using services like **Virustotal**.
Finally, the extracted features are organized into a structured format and stored in a new CSV file for further analysis and classification purposes.

3.3 Machine Learning Algorithms

Machine learning algorithms are computational procedures that enable computers to learn from data and make predictions or decisions without being explicitly programmed. These algorithms are a key component of artificial intelligence (AI) systems and are used in various applications across industries. Machine learning algorithms can be broadly categorized into different types based on their learning approach and the tasks they perform.

3.3.1 Random Forest

A Random Forest model is an ensemble learning method primarily used for classification tasks in supervised machine learning. It comprises multiple decision trees, each trained on random subsets of the training data and features, to reduce overfitting and enhance model diversity. Decision trees segment the feature space recursively, making predictions based on feature values. Through bagging, each tree is trained on slightly different data subsets, further reducing variance. Random Forests also employ random feature selection at each split to prevent the dominance of particular features. In classification, the final prediction is determined by a majority vote among individual trees. Renowned for their robustness and ability to handle high-dimensional data, Random Forest models find widespread applications across various domains, including finance, healthcare, and image recognition.
Each tree is trained on a bootstrap sample of the dataset, and at each split, a random subset of features is considered. The final prediction is made by averaging or voting over all trees. Mathematically, the prediction $\hat{y}$ for a new instance x in a Random Forest with B trees can be

represented as:

$$\hat{y} = \frac{1}{B} \sum_{i=1}^{B} f_i(x)$$

3.3.2 XgBoost

XGBoost, or Extreme Gradient Boosting, stands as a formidable machine learning algorithm effectivent in regression and classification tasks. Operating within the gradient boosting framework, XGBoost sequentially incorporates a series of weak learners, often decision trees, to iteratively refine predictions. A key aspect of XGBoost lies in its utilization of gradient descent optimization to minimize an objective function, comprised of both a loss function, measuring the disparity between actual and predicted values, and a regularization term, which penalizes model complexity.

$$\sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where L is the loss function and is the regularization term. The objective function is mathematically represented as the sum of the loss function and the regularization term. Through this iterative process, XGBoost iteratively minimizes the error while preventing overfitting through regularization. Furthermore, XGBoost provides insights into feature importance, aiding in model interpretability. Its robustness, efficiency, and ability to handle complex datasets make it a favored choice across various domains, from machine learning competitions to practical applications.

3.3.3 CatBoost

CatBoost is a state-of-the-art gradient boosting algorithm specifically designed to handle categorical features efficiently. Developed by Yandex, CatBoost stands out for its robustness, scalability, and ability to deliver high-quality predictions with minimal data preprocessing.

One of the key strengths of CatBoost lies in its ability to handle categorical variables seamlessly without the need for extensive feature engineering or manual encoding. Unlike traditional gradient boosting algorithms, which require categorical features to be converted into numerical representations through techniques like one-hot encoding, CatBoost can directly process categorical features in their original form. This capability not only simplifies the preprocessing pipeline but also helps retain valuable information encoded within categorical variables.

CatBoost employs a novel approach to gradient boosting, utilizing an innovative algorithm for decision tree construction known as "ordered boosting." This technique incorporates the natural order of categorical variables during the tree-building process, thereby improving model performance and reducing overfitting. Additionally, CatBoost employs advanced regularization techniques such as ordered boosting, random permutations, and learning rate annealing to further enhance model generalization and prevent overfitting.

Another notable feature of CatBoost is its support for GPU acceleration, enabling faster model training and inference on large datasets. By harnessing the computational power of GPUs, CatBoost can significantly reduce training times and improve scalability, making it well-suited for handling large-scale datasets and real-world applications.

Furthermore, CatBoost offers built-in support for handling missing values, automatic feature selection, and model interpretability through feature importance analysis. These features provide users with valuable insights into the underlying patterns driving predictions, facilitating model understanding and debugging.

In summary, CatBoost represents a powerful gradient boosting algorithm tailored for handling categorical data efficiently. With its robust performance, support for GPU acceleration, and built-in capabilities for handling categorical variables, CatBoost is a valuable tool for a wide range of machine learning tasks, including classification, regression, and ranking.

3.4 Deep Learning Algorithms

Deep learning algorithms are a subset of machine learning methods that are designed to automatically learn hierarchical representations of data through the use of neural networks with multiple layers. These algorithms have revolutionized various fields such as computer vision, natural language processing, speech recognition, and more, by enabling complex pattern recognition and feature extraction from raw data. Unlike traditional machine learning algorithms that require handcrafted features, deep learning algorithms can automatically learn relevant features from the data itself, making them highly flexible and powerful.

3.4.1 LSTM

Long Short-Term Memory (LSTM) is an advanced variant of recurrent neural networks (RNNs), engineered to address the challenges of learning and retaining temporal dependencies in sequential data. At its core, an LSTM unit comprises a memory cell capable of storing information over extended time intervals, enabling the network to effectively capture long-range dependencies. Unlike traditional RNNs, LSTM networks introduce specialized gating mechanisms that regulate the flow of information within the cell. These gates, including the input, forget, and output gates, control the updating and retrieval of information, thereby mitigating the issues of vanishing or exploding gradients encountered during training. Through careful management of information flow, LSTMs can selectively remember or forget past inputs, allowing them to discern relevant patterns and retain important context over prolonged sequences.

The robust memory capabilities of LSTM networks make them well-suited for a wide range of sequential data tasks, including natural language processing, time series analysis, and speech recognition. In natural language processing, for instance, LSTMs excel at understanding and generating coherent sentences by capturing dependencies between words spread across long distances. Similarly, in time series forecasting, LSTMs can effectively model intricate patterns and dependencies in historical data, enabling accurate predictions of future trends. The versatility and effectiveness of LSTM networks stem from their ability to balance short-term memory for capturing immediate context with long-term memory for preserving overarching patterns, making them a cornerstone technology in sequential data analysis and prediction.

Mathematically, the computations in an LSTM cell at time step t can be represented by the following formulas:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}C_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}C_{t-1} + b_f) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}C_t + b_o) \\
\tilde{C}_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
h_t &= o_t \odot \tanh(C_t)
\end{aligned}
$$

3.4.2 CNN-LSTM

A Convolutional Neural Network - Long Short-Term Memory (CNN-LSTM) architecture represents a hybrid model that amalgamates the strengths of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. This unique fusion is particularly adept at analyzing sequential data with spatial features, making it well-suited for a variety of tasks such as image captioning, video classification, and time series analysis.

At its core, the CNN-LSTM architecture integrates CNN layers for feature extraction with LSTM layers for sequential learning. The CNN layers are responsible for capturing spatial patterns and local dependencies within the input data, leveraging filters to convolve across the input and extract relevant features. This feature extraction process is crucial for identifying meaningful patterns and structures within the data, especially in tasks involving images or sequences with spatial dependencies.

Following the CNN layers, the output is passed onto LSTM layers, which excel in capturing long-term dependencies and temporal dynamics within sequential data. The LSTM architecture, composed of memory cells and gates, enables the model to retain and selectively update information over time, facilitating the learning of complex temporal patterns and relationships.

By combining CNNs and LSTMs, the CNN-LSTM architecture capitalizes on the strengths of both models: the CNN's ability to extract spatial features and the LSTM's proficiency in modeling temporal dependencies. This synergy allows the model to effectively capture both spatial and temporal information, making it highly versatile and well-suited for tasks where both aspects are crucial for accurate prediction or classification.

In the context of URL classification and threat detection, the CNN-LSTM architecture can be leveraged to analyze sequences of URL features while simultaneously capturing spatial patterns within the data. This enables the model to effectively classify URLs based on their characteristics and identify potential threats or malicious activity. Overall, the CNN-LSTM architecture represents a powerful tool in the arsenal of deep learning techniques, offering enhanced capabilities for analyzing complex sequential data with spatial dependencies.

3.5 Explainable AI

Explainable AI (XAI) is a crucial aspect of machine learning models, aiming to provide insights into how predictions are made, particularly in complex black-box models. SHAP (**SH**apley **A**dditive ex**P**lanations) is a popular technique in XAI that helps to interpret the output of machine learning models by attributing the contribution of each feature to the model's predictions. In the provided function `shap_analysis`, the SHAP library is utilized to compute SHAP values for the learning models. By employing a `TreeExplainer` specific to the model, SHAP values are computed, revealing the impact of each feature on individual predictions. These SHAP values are then visualized using a SHAP summary plot, providing a comprehensive overview of feature importance. The summary plot displays the average magnitude of SHAP values for each feature, allowing users to identify the most influential features in the model's decision-making process. This visualization aids in understanding the underlying mechanisms driving the model's predictions and enhances its interpretability, which is crucial for building trust and deploying AI systems in real-world applications.

3.6 Gemini API

The Google's Gemini API model, also known as GenAI, provides a powerful framework for leveraging artificial intelligence capabilities in various applications. In our model, we utilize the Gemini API to enhance the analysis provided by the SHAP (SHapley Additive ex-

Planations) graph and enable interaction with users. By configuring the GenAI module with an API key, we gain access to advanced generative models like the 'gemini-pro-vision.' This model allows us to generate content based on input queries and additional contextual information. In our implementation, we incorporate the SHAP graph, which visually represents the impact of features on model predictions, as part of the input to the generative model. By combining the user's query with the SHAP graph, we create a comprehensive context for content generation. The Gemini API model then processes this input and generates insightful analysis or responses tailored to the user's query and the information provided by the SHAP graph. This integration of AI-driven content generation with SHAP-based analysis enriches the user experience and facilitates more informative and interactive interactions within our model.

3.7 Plugin

The plugin model operates seamlessly to provide users with a comprehensive analysis of URLs and interactive query answering. Upon user interaction with the plugin interface, the frontend JavaScript code orchestrates the process by sending the URL input to the Flask backend server for analysis. The backend server employs a robust learning algorithm to classify the content of the URL accurately. This classification result is then returned to the frontend, where it is displayed to the user. If a category is successfully obtained, the frontend initiates an AI-driven query answering session using Google's Gemini API model. The user is prompted to input queries, which are then sent to the backend server for processing. The backend server utilizes the Gemini API model, enriched with SHAP graph information derived from the URL analysis, to generate contextually relevant responses to the user's queries. These responses are then returned to the frontend and displayed to the user in real-time.

Chapter 4

DESIGN & IMPLEMENTATION

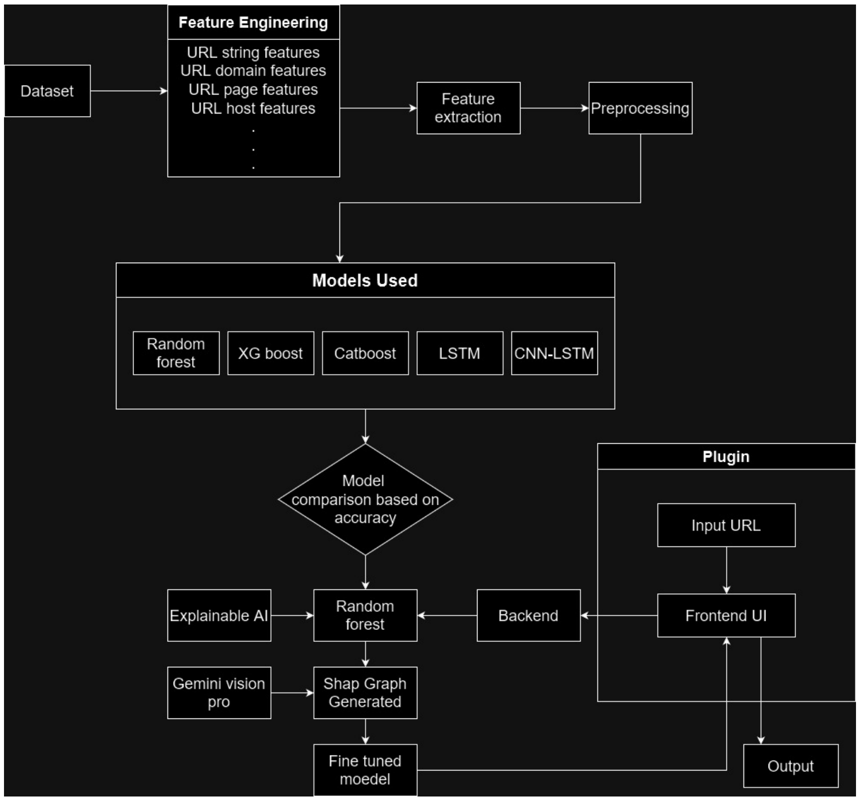


Figure 4.1: Model Flowchart

In the implementation phase, we initiated the process by extracting features from the comprehensive ISCX-2016 dataset, encompassing URLs classified into various categories such as benign, malware, spam, phishing, and defacement. Employing Python scripting, we meticulously crafted code to extract pertinent features from each dataset file. Subsequently, these extracted features were amalgamated into a unified CSV file, laying the groundwork for subsequent analysis and utilization.

A class named URLFeaturizer was implemented to encapsulate the logic for extracting various features from a URL. The following are the URL features extracted along with their relevance:

URL String Features

- **Entropy**: Measures the randomness of characters in the URL, indicating potential obfuscation.
- **IP**: Checks if the URL contains an IP address, suggesting a direct connection rather than a domain name.
- **Number of Digits**: Counts the digits in the URL, which may indicate dynamic parameters or encoded data.
- **URL Length**: Determines the length of the URL, potentially indicating complexity or suspiciously long URLs.
- **Number of Parameters**: Counts the number of parameters in the URL, revealing potential data transmission or tracking mechanisms.
- **Number of Fragments**: Counts the number of fragments in the URL, indicating potential navigation within the page.
- **Number of Subdomains**: Counts the number of subdomains in the URL, indicating potential subdomains or nesting within the domain.
- **Domain Extension**: Extracts the domain extension from the URL, indicating the type of organization or country code.
- **Check word server/client**: Check for existence of keywords "server" or "client" in domain.
- **Presence of HTTP/HTTPS**: Checks if the URL starts with "http://" or "https://", indicating secure or insecure connections.

URL Domain Features

- **Days Since Registration**: Calculates the number of days since the domain registration, providing insight into the age of the domain.

- **Days Since Expiration**: Calculates the number of days until the domain expiration, indicating potential short-term domains or expiration issues.

URL Page Features

- **Body Length**: Determines the length of the webpage body, indicating the complexity or richness of content.
- **Number of Titles**: Counts the number of titles in the webpage, indicating the structure or organization of the content.
- **Number of Images**: Counts the number of images in the webpage, indicating visual richness or potential for phishing.
- **Number of Links**: Counts the number of links in the webpage, indicating navigation options or potential for redirection.
- **Script Length**: Determines the length of scripts in the webpage, indicating potential for dynamic content or scripting attacks.
- **Special Characters**: Counts the number of special characters in the webpage, indicating potential for obfuscation or scripting.
- **Script to Special Characters Ratio**: Calculates the ratio of script length to special characters count, indicating potential for code injection or malicious scripts.
- **Script to Body Ratio**: Calculates the ratio of script length to body length, indicating the prominence of scripting in the webpage.
- **Body to Special Character Ratio**: Calculates the ratio of body length to special characters count, indicating potential for content obfuscation.

Additional Features

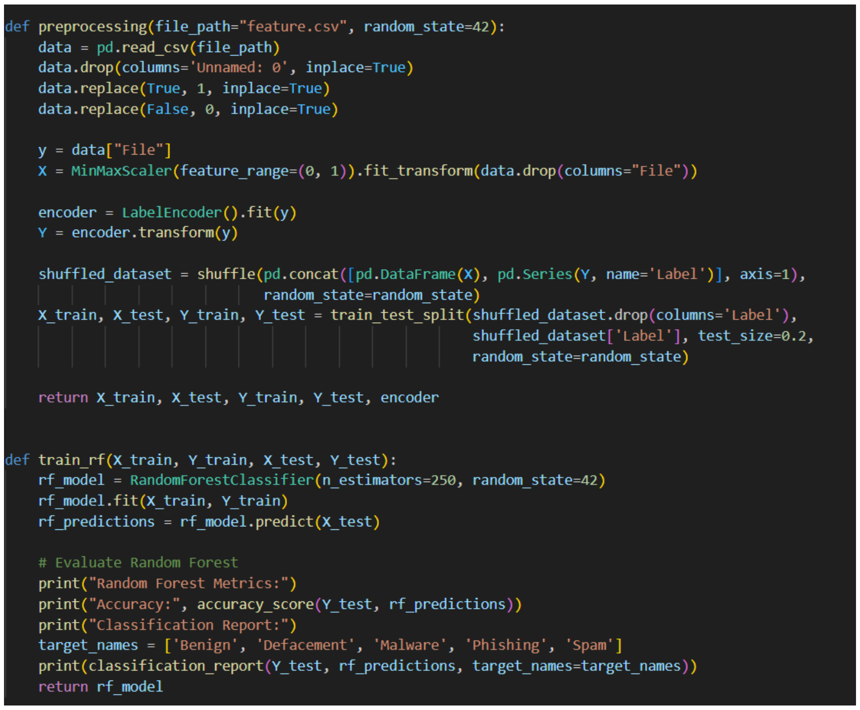
- **Presence of Top-Level Domains (TLD)**: Checks for the presence of TLDs in the URL, indicating potential spoofing or phishing attempts.

- **Count of TLDs**: Counts the number of TLDs in the URL, providing insight into the complexity of domain naming.
- **Presence of Keywords**: Checks for the presence of keywords like "server" or "client" in the URL, indicating potential for specific server-related activities.
- **Count of Name Servers**: Counts the number of resolved name servers, indicating potential for domain resolution issues or distributed infrastructure.
- **Count of Mail Servers**: Counts the number of resolved mail servers.
- **Presence of SSL Certificate**: Checks if the SSL certificate is valid, indicating secure communication channels.
- **Count of Redirects**: Counts the number of redirects in the URL, indicating potential for redirection attacks or navigation complexity.
- **Autonomous System Number (ASN)**: Retrieves the ASN associated with the IP, providing insight into the network infrastructure.
- **PTR (Pointer Record)**: Retrieves the PTR associated with the IP, indicating potential reverse DNS resolution or domain associations.
- **Presence on Real-time Blackhole List (RBL)**: Checks if the domain is listed on RBL, indicating potential for spam or blacklisting.
- **Blacklisting Check**: Checks if the URL or domain is listed on blacklists like VirusTotal, indicating potential for malicious activity or reputation issues.

After consolidating the extracted features into a unified file, they undergo preprocessing and evaluation with multiple learning models to identify the optimal model for subsequent utilization.

The **preprocessing** function is common to all and performs data preprocessing tasks for classification. It reads the file and removes any unnecessary columns, scales the feature values to a range between 0 and 1 using Min-Max scaling, encodes the categorical labels using Label Encoding, shuffles the dataset, splits it into training and testing sets, and finally returns

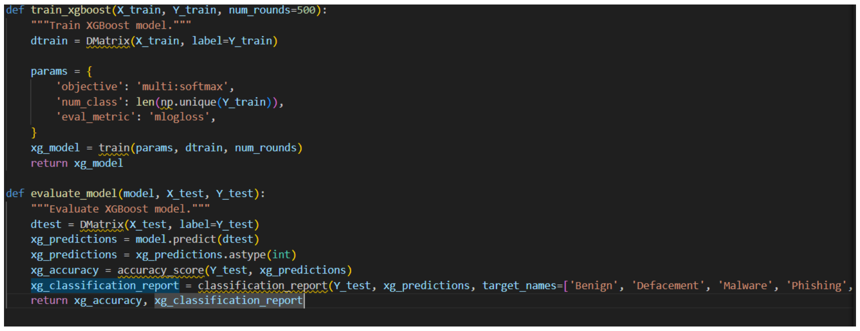
the preprocessed training and testing data along with the label encoder.
We begin by using the **Random Forest Algorithm** The **train_rf** function trains a Random Forest classifier using the preprocessed training data. It initializes a Random Forest model with 250 estimators, fits it to the training data, makes predictions on the test data, and evaluates the model's performance using accuracy score and classification report metrics, including precision, recall, and F1-score for each class (Benign, Defacement, Malware, Phishing, Spam). Finally, it returns the trained Random Forest model. It then takes the label encoder, feature scaler, and trained model as inputs and saves them to designated file paths using NumPy and pickle serialization. It saves the label encoder's classes to a NumPy file, serializes the scaler and model objects using pickle, and stores them as binary files.



```python
def preprocessing(file_path="feature.csv", random_state=42):
    data = pd.read_csv(file_path)
    data.drop(columns='Unnamed: 0', inplace=True)
    data.replace(True, 1, inplace=True)
    data.replace(False, 0, inplace=True)

    y = data["File"]
    X = MinMaxScaler(feature_range=(0, 1)).fit_transform(data.drop(columns="File"))

    encoder = LabelEncoder().fit(y)
    Y = encoder.transform(y)

    shuffled_dataset = shuffle(pd.concat([pd.DataFrame(X), pd.Series(Y, name='Label')], axis=1),
                               random_state=random_state)
    X_train, X_test, Y_train, Y_test = train_test_split(shuffled_dataset.drop(columns='Label'),
                                                        shuffled_dataset['Label'], test_size=0.2,
                                                        random_state=random_state)

    return X_train, X_test, Y_train, Y_test, encoder


def train_rf(X_train, Y_train, X_test, Y_test):
    rf_model = RandomForestClassifier(n_estimators=250, random_state=42)
    rf_model.fit(X_train, Y_train)
    rf_predictions = rf_model.predict(X_test)

    # Evaluate Random Forest
    print("Random Forest Metrics:")
    print("Accuracy:", accuracy_score(Y_test, rf_predictions))
    print("Classification Report:")
    target_names = ['Benign', 'Defacement', 'Malware', 'Phishing', 'Spam']
    print(classification_report(Y_test, rf_predictions, target_names=target_names))
    return rf_model
```

Figure 4.2: Preprocessing & Random Forest Model

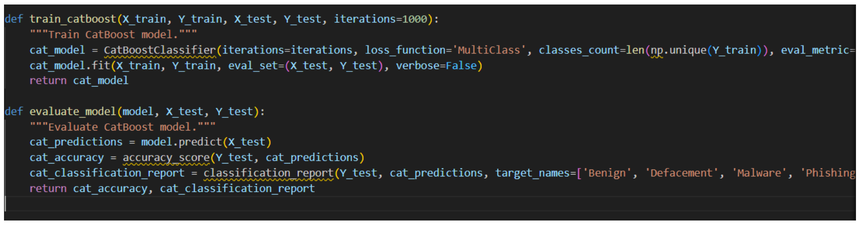
Next we try out with **XgBoost Algorithm**, The **train_xgboost** function trains an XGBoost model using the provided training features X_{train} and labels Y_{train}. It initializes a DMatrix object for the training data, sets parameters for multi-class classification, and trains the model for 500 rounds. The trained XGBoost model is returned. The **evaluate_model**gives a trained XGBoost model *model*, test features X_{test}, and corresponding labels Y_{test}, this function evaluates the model's performance. It creates a DMatrix object for the test data, predicts labels using the model, and calculates the accuracy of the predictions.



```
def train_xgboost(X_train, Y_train, num_rounds=500):
    """Train XGBoost model."""
    dtrain = DMatrix(X_train, label=Y_train)

    params = {
        'objective': 'multi:softmax',
        'num_class': len(np.unique(Y_train)),
        'eval_metric': 'mlogloss',
    }
    xg_model = train(params, dtrain, num_rounds)
    return xg_model

def evaluate_model(model, X_test, Y_test):
    """Evaluate XGBoost model."""
    dtest = DMatrix(X_test, label=Y_test)
    xg_predictions = model.predict(dtest)
    xg_predictions = xg_predictions.astype(int)
    xg_accuracy = accuracy_score(Y_test, xg_predictions)
    xg_classification_report = classification_report(Y_test, xg_predictions, target_names=['Benign', 'Defacement', 'Malware', 'Phishing',
    return xg_accuracy, xg_classification_report
```

Figure 4.3: XgBoost Model

Moving onto **Catboost Algorithm**, The **train_catboost** function trains a CatBoost model using the provided training features X_{train} and labels Y_{train}, along with the test features X_{test} and labels Y_{test}. It initializes a CatBoostClassifier with specified parameters such as the number of iterations, loss function, and evaluation metric. The model is then fitted to the training data, and its performance is evaluated on the test set using accuracy and a classification report. The trained CatBoost model is returned. The **evaluate_model** gives a trained CatBoost model *model*, test features X_{test}, and corresponding labels Y_{test}, this function evaluates the model's performance. It makes predictions on the test set using the trained model, calculates the accuracy of the predictions, and generates a classification report based on the predicted and true labels. The accuracy and classification report are returned as outputs.

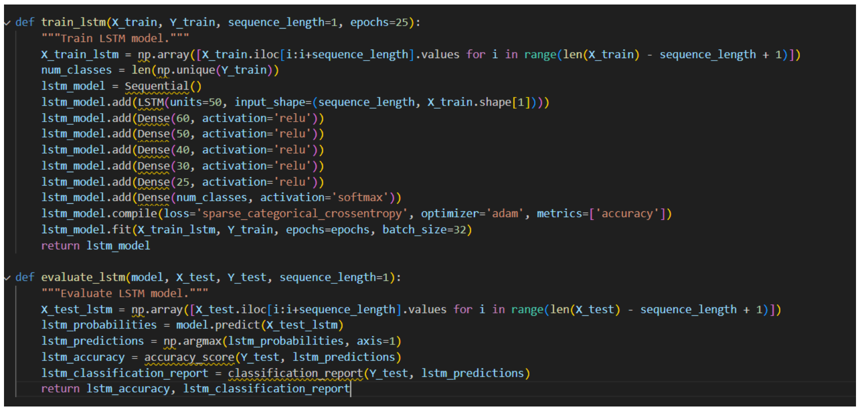


```
def train_catboost(X_train, Y_train, X_test, Y_test, iterations=1000):
    """Train CatBoost model."""
    cat_model = CatBoostClassifier(iterations=iterations, loss_function='MultiClass', classes_count=len(np.unique(Y_train)), eval_metric=
    cat_model.fit(X_train, Y_train, eval_set=(X_test, Y_test), verbose=False)
    return cat_model

def evaluate_model(model, X_test, Y_test):
    """Evaluate CatBoost model."""
    cat_predictions = model.predict(X_test)
    cat_accuracy = accuracy_score(Y_test, cat_predictions)
    cat_classification_report = classification_report(Y_test, cat_predictions, target_names=['Benign', 'Defacement', 'Malware', 'Phishing
    return cat_accuracy, cat_classification_report
```

Figure 4.4: Catboost Model

Shifting our focus to Deep Learning models we have used **LSTM**, The **train_lstm** function trains an LSTM model using the provided training features X_{train} and labels Y_{train}, with an optional parameter sequence_length specifying the length of the sequences as 1 is used for training, and epochs specifying the number of training epochs to 25. It reshapes the input features into sequences of the specified length, constructs the 7 layer LSTM model architecture with specified units and activation functions for the hidden layers and output layer, compiles the model with the appropriate loss function and optimizer, and fits the model to the training data. The trained LSTM model is returned.

The **evaluate_lstm** gives a trained LSTM model *model*, test features X_{test}, and corresponding labels Y_{test}, this function evaluates the model's performance. It reshapes the test features into sequences of the specified length, makes predictions on the test set using the trained model, calculates the accuracy of the predictions, and generates a classification report based on the predicted and true labels. The accuracy and classification report are returned as outputs.

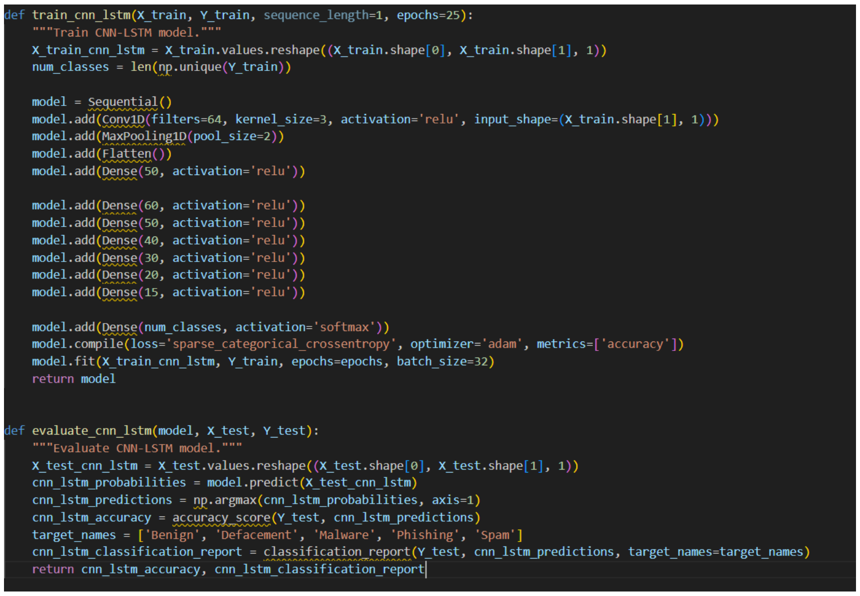


```
def train_lstm(X_train, Y_train, sequence_length=1, epochs=25):
    """Train LSTM model."""
    X_train_lstm = np.array([X_train.iloc[i:i+sequence_length].values for i in range(len(X_train) - sequence_length + 1)])
    num_classes = len(np.unique(Y_train))
    lstm_model = Sequential()
    lstm_model.add(LSTM(units=50, input_shape=(sequence_length, X_train.shape[1])))
    lstm_model.add(Dense(60, activation='relu'))
    lstm_model.add(Dense(50, activation='relu'))
    lstm_model.add(Dense(40, activation='relu'))
    lstm_model.add(Dense(30, activation='relu'))
    lstm_model.add(Dense(25, activation='relu'))
    lstm_model.add(Dense(num_classes, activation='softmax'))
    lstm_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    lstm_model.fit(X_train_lstm, Y_train, epochs=epochs, batch_size=32)
    return lstm_model

def evaluate_lstm(model, X_test, Y_test, sequence_length=1):
    """Evaluate LSTM model."""
    X_test_lstm = np.array([X_test.iloc[i:i+sequence_length].values for i in range(len(X_test) - sequence_length + 1)])
    lstm_probabilities = model.predict(X_test_lstm)
    lstm_predictions = np.argmax(lstm_probabilities, axis=1)
    lstm_accuracy = accuracy_score(Y_test, lstm_predictions)
    lstm_classification_report = classification_report(Y_test, lstm_predictions)
    return lstm_accuracy, lstm_classification_report
```

Figure 4.5: LSTM Model

Then we used **CNN-LSTM**. The **train_cnn_lstm** function trains a Convolutional Neural Network (CNN) followed by a Long Short-Term Memory (LSTM) model using the provided training features X_{train} and labels Y_{train}, with optional parameters sequence_length specifying the length of the sequences as 1 which is used for training and epochs specifying the number of training epochs to 25. It reshapes the input features to be suitable for the CNN-LSTM architecture, constructs the model with convolutional layers followed by dense layers, compiles the model with the appropriate loss function and optimizer, and fits the model to the training data. The trained CNN-LSTM model is returned.

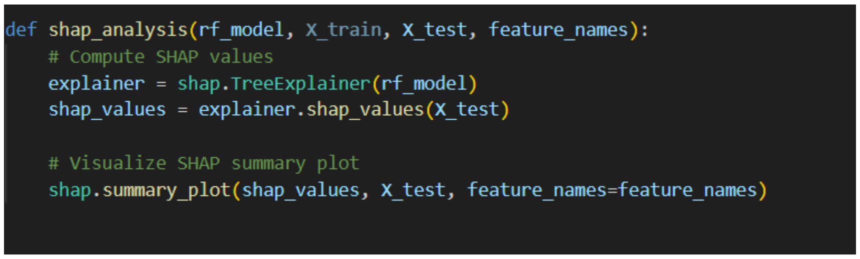
The **evaluate_cnn_lstm** Gives a trained CNN-LSTM model *model*, test features X_{test}, and corresponding labels Y_{test}, this function evaluates the model's performance. It reshapes the test features to match the input shape expected by the model, makes predictions on the test set using the trained model, calculates the accuracy of the predictions, and generates a classification report based on the predicted and true labels. The accuracy and classification report are returned as outputs.



```python
def train_cnn_lstm(X_train, Y_train, sequence_length=1, epochs=25):
    """Train CNN-LSTM model."""
    X_train_cnn_lstm = X_train.values.reshape((X_train.shape[0], X_train.shape[1], 1))
    num_classes = len(np.unique(Y_train))

    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))

    model.add(Dense(60, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(40, activation='relu'))
    model.add(Dense(30, activation='relu'))
    model.add(Dense(20, activation='relu'))
    model.add(Dense(15, activation='relu'))

    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X_train_cnn_lstm, Y_train, epochs=epochs, batch_size=32)
    return model


def evaluate_cnn_lstm(model, X_test, Y_test):
    """Evaluate CNN-LSTM model."""
    X_test_cnn_lstm = X_test.values.reshape((X_test.shape[0], X_test.shape[1], 1))
    cnn_lstm_probabilities = model.predict(X_test_cnn_lstm)
    cnn_lstm_predictions = np.argmax(cnn_lstm_probabilities, axis=1)
    cnn_lstm_accuracy = accuracy_score(Y_test, cnn_lstm_predictions)
    target_names = ['Benign', 'Defacement', 'Malware', 'Phishing', 'Spam']
    cnn_lstm_classification_report = classification_report(Y_test, cnn_lstm_predictions, target_names=target_names)
    return cnn_lstm_accuracy, cnn_lstm_classification_report
```

Figure 4.6: CNN-LSTM Model

After identifying the model with the best accuracy from our experiments, we will leverage it for further tasks in our project. This model represents our best-performing solution and will be utilized for various downstream applications. With the selected model in hand, we can proceed to incorporate eXplainable Artificial Intelligence (XAI) techniques to gain insights into the model's decision-making process and enhance its interpretability. Our approach is to integrating XAI into our model by creating SHAP (SHapley Additive exPlanations) graphs. SHAP is a powerful technique for interpreting black-box models by explaining individual predictions. By generating SHAP values for each feature in our dataset, we can visualize the impact of different features on the model's predictions. This not only provides valuable insights into how the model makes decisions but also helps identify which features are the most influential in driving those decisions.



```python
def shap_analysis(rf_model, X_train, X_test, feature_names):
    # Compute SHAP values
    explainer = shap.TreeExplainer(rf_model)
    shap_values = explainer.shap_values(X_test)

    # Visualize SHAP summary plot
    shap.summary_plot(shap_values, X_test, feature_names=feature_names)
```

Figure 4.7: SHAP Model

Utilizing Google's Gemini Vision Pro LLM (Large Language Model) in conjunction with the SHAP graph generated from our model represents a sophisticated approach to building an interactive chatbot interface. By fine-tuning Gemini Vision Pro LLM with the SHAP graph, we can provide users with more meaningful and interpretable responses:

- **API Call to Google**: When a request is received from the frontend, the Flask server initiates an API call to Google using the provided API key. This API call can leverage various Google services, such as Gemini Vision Pro LLM, for natural language processing tasks. The API response contains the processed data or insights generated by Google's services.
- **Model Inference**: In parallel to the API call to Google, the Flask server performs inference using the machine learning models hosted on the backend. This includes processing the input data received from the frontend, making predictions using the models, and generating classification responses based on the model with the best accuracy.
- **Response Integration**: Once the API response from Google and the model's classification response are obtained, the Flask server integrates these responses into a single coherent response. This combined response is then sent back to the frontend for display or further processing.

A JavaScript code orchestrates a seamless user experience on the frontend. Upon the user clicking the "classify" button, the `classifyUrl` function is invoked, extracting the URL input. This URL is then dispatched to the backend server via a POST request to a designated endpoint (`/classify`). The backend processes the URL, determines its category, and returns the result, which is dynamically displayed on the webpage. If a category is obtained, the code proceeds to initiate interaction with an AI model by invoking the `startAiInteraction` function. This function prompts the user for queries, sends them to the backend server via POST requests to the `/query` endpoint, and displays the AI-generated responses in real-time. This elegant integration seamlessly blends user input, backend processing, and AI interaction to deliver a fluid and intuitive experience to the user.

4.1 Technology Stack

4.1.1 Python

We employed a robust technology stack centered around Python and its diverse libraries. Python served as the backbone of our project, providing a versatile and intuitive programming environment. We leveraged the datetime library for handling date and time operations, facilitating efficient timestamp management throughout our data processing pipeline. The whois library enabled us to extract and analyze domain registration information, enhancing our URL feature extraction process with valuable domain insights. Utilizing the math library, we conducted various mathematical computations and calculations, ensuring accuracy and precision in our classification algorithms. The powerful pandas library empowered us to manipulate and analyze large datasets effortlessly, facilitating data preprocessing and exploration tasks. Additionally, we utilized the pyquery library for web scraping and parsing HTML content, enabling us to extract relevant information from web pages for training and testing our classification models. Together, these libraries formed a comprehensive technology stack that underpinned our URL classification project, enabling us to efficiently process, analyze, and classify URLs with confidence and accuracy.

4.1.2 Scikit-learn

We harnessed the capabilities of the scikit-learn library, commonly referred to as sklearn, to implement machine learning algorithms and perform various tasks essential to our project's success. Scikit-learn provided us with a rich set of tools and functionalities for data preprocessing, model training, evaluation, and inference. With its user-friendly interface and extensive documentation, scikit-learn enabled us to effortlessly preprocess our dataset, including feature scaling, encoding categorical variables, and handling missing values. We leveraged its wide range of supervised and unsupervised learning algorithms, such as decision trees, random forests, support vector machines, and k-nearest neighbors, to train robust classification models capable of distinguishing between benign and malicious URLs. Additionally, scikit-learn facilitated model evaluation through its comprehensive suite of metrics, enabling us to assess the performance of our classifiers in terms of accuracy, precision, recall, and F1 score. Its seamless integration with other Python libraries and frameworks streamlined our workflow, allowing us to iterate rapidly and experiment with different algorithms and parameters to optimize model performance. Overall, scikit-learn played a pivotal role in our project, empowering us to build and deploy effective URL classification solutions with ease and efficiency.

4.1.3 Keras

Keras is a Python-based open-source deep learning library designed for simplicity and flexibility in building neural networks. Developed by François Chollet, it offers a high-level interface that abstracts away complexity, allowing users to focus on model design and experimentation rather than low-level implementation details. With its modular structure and consistent API, Keras facilitates the construction of various types of neural networks, including convolutional and recurrent networks, while seamlessly integrating with popular backend frameworks like TensorFlow. Its popularity stems from its ease of use, extensive documentation, and interoperability, making it suitable for both beginners and experienced practitioners in the field of deep learning.

4.1.4 Tensorflow

TensorFlow is an open-source machine learning framework developed by Google Brain, facilitating the creation, training, and deployment of machine learning models, particularly deep neural networks. Its core representation of computations as data flow graphs enables efficient execution, with nodes representing operations and edges symbolizing the flow of data in tensor form, allowing for automatic differentiation and optimization techniques like gradient descent during model training.This framework provides a comprehensive ecosystem of tools and libraries, including high-level APIs like Keras for neural network development, TensorFlow Extended (TFX) for complete machine learning pipelines, and TensorFlow Serving for deploying models in production environments. With broad community support, extensive documentation, and performance optimizations, TensorFlow has emerged as one of the most prevalent frameworks for machine learning and deep learning applications

4.1.5 SHAP

The integration of SHAP (SHapley Additive exPlanations) in our project plays a crucial role in enhancing the interpretability of our machine learning-based URL Classification system. SHAP values provide insights into how individual features contribute to classification decisions, bridging the gap between model accuracy and transparency. By leveraging SHAP's visualizations and summaries, we can effectively communicate classification results to stakeholders, fostering trust and enabling better-informed decision-making.Furthermore, the incorporation of Explainable AI (XAI) techniques such as SHAP not only improves model debugging and refinement but also ensures that our URL classification system meets specific domain constraints and requirements. This strategic integration not only enhances transparency and reliability but also underscores our commitment to building a robust and trustworthy ML-driven solution. Overall, SHAP's role in our XAI strategy contributes significantly to the success and acceptance of our URL Classification system by stakeholders and end-users alike.

4.1.6 Gemini

The inclusion of Gemini AI represents a significant advancement in enhancing the interpretability and user interaction of our classification system. Gemini AI, developed by Google, stands at the forefront of Explainable Artificial Intelligence (XAI) technology, offering an intuitive and interactive interface that allows users to gain deeper insights into model predictions and decision-making processes. By leveraging state-of-the-art techniques such as SHAP (SHapley Additive exPlanations), Gemini AI provides users with a transparent view of feature importance, enabling them to understand how individual features contribute to the model's predictions. The interactive nature of Gemini AI empowers users to ask specific questions about individual predictions and receive detailed explanations in return, fostering a deeper understanding of the model's behavior and boosting trust and confidence in its capabilities.

Chapter 5

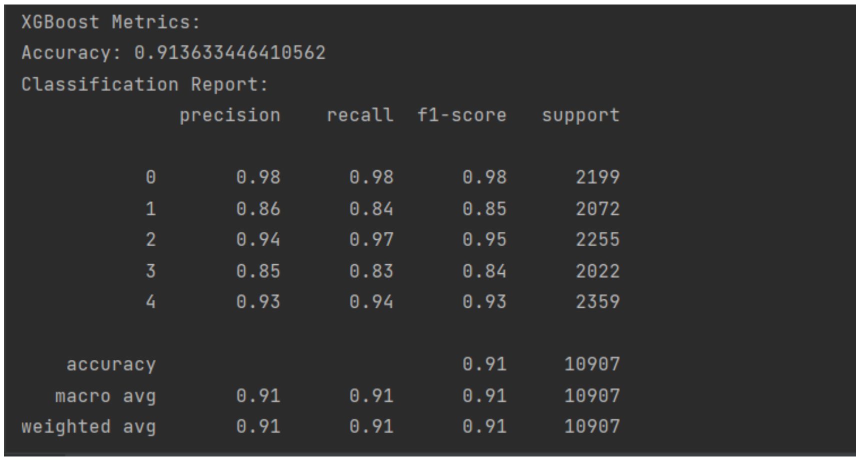
RESULT & DISCUSSION

In our comparative analysis of learning models, we examined six diverse algorithms, each offering distinct strengths and capabilities. XGBoost, a gradient boosting algorithm, demonstrated robust performance and adaptability through its ensemble learning approach. CatBoost, specialized in handling categorical data, exhibited competitive accuracy and showcased its effectiveness in real-world applications. Random Forest, a versatile ensemble method, demonstrated resilience to overfitting and high interpretability, making it well-suited for exploratory analysis and feature importance ranking. LSTM, a recurrent neural network architecture, excelled in capturing sequential patterns, making it ideal for time-series data like URL features. CNN-LSTM, which combines convolutional and recurrent layers, effectively utilized both spatial and sequential aspects of data, proving valuable for structured input data. Lastly, the ensemble of models with SHAP graph integration underscored the importance of interpretability and the potential for leveraging external insights to enhance classification performance. Through this comprehensive comparison, we gained insights into the diverse landscape of learning algorithms and their applicability across various domains and use cases.

Further elaboration and detailed analysis of the models are expounded upon in subsequent sections of this chapter.

5.0.1 XgBoost

XGBoost, an implementation of gradient boosting, stands out for its efficiency and scalability in handling large datasets and complex classification tasks. In our project, XGBoost plays a pivotal role in enhancing the accuracy and robustness of our classification system. By iteratively training weak learners to correct the errors made by preceding models, XGBoost iteratively builds a strong ensemble model capable of achieving high predictive performance. With an impressive accuracy rate of 91%, XGBoost demonstrates its effectiveness in accurately classifying URLs into their respective categories, providing valuable insights for cybersecurity threat detection and mitigation strategies. Its ability to handle diverse feature sets, manage class imbalances, and optimize hyperparameters makes XGBoost a powerful tool for our project's classification tasks.

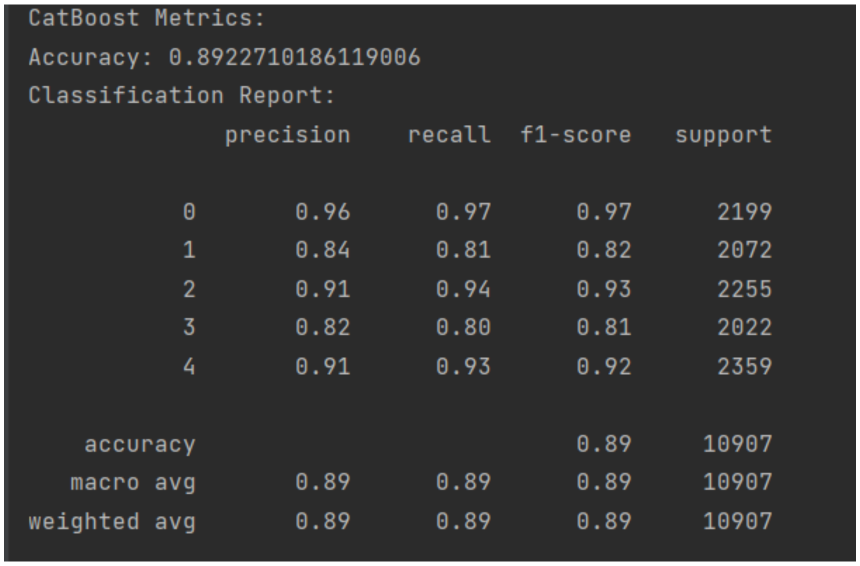


```
XGBoost Metrics:
Accuracy: 0.913633446410562
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2199
           1       0.86      0.84      0.85      2072
           2       0.94      0.97      0.95      2255
           3       0.85      0.83      0.84      2022
           4       0.93      0.94      0.93      2359

    accuracy                           0.91     10907
   macro avg       0.91      0.91      0.91     10907
weighted avg       0.91      0.91      0.91     10907
```

Figure 5.1: XgBoost Classification Report

5.0.2 Catboost

CatBoost, a gradient boosting algorithm developed by Yandex, offers notable advantages in handling categorical features and robustness to overfitting. In our project, CatBoost plays a crucial role in augmenting the accuracy and reliability of our classification system. By incorporating an innovative approach to handling categorical variables through the implementation of an ordered boosting scheme, CatBoost effectively addresses the challenges posed by such features in our dataset. With an impressive accuracy rate of 89%, CatBoost demonstrates its capability to accurately classify URLs into their corresponding categories, thereby providing valuable insights for identifying and mitigating cybersecurity threats. Its ability to automatically handle missing data, reduce the need for extensive data preprocessing, and optimize model performance makes CatBoost a valuable addition to our project's classification toolkit.

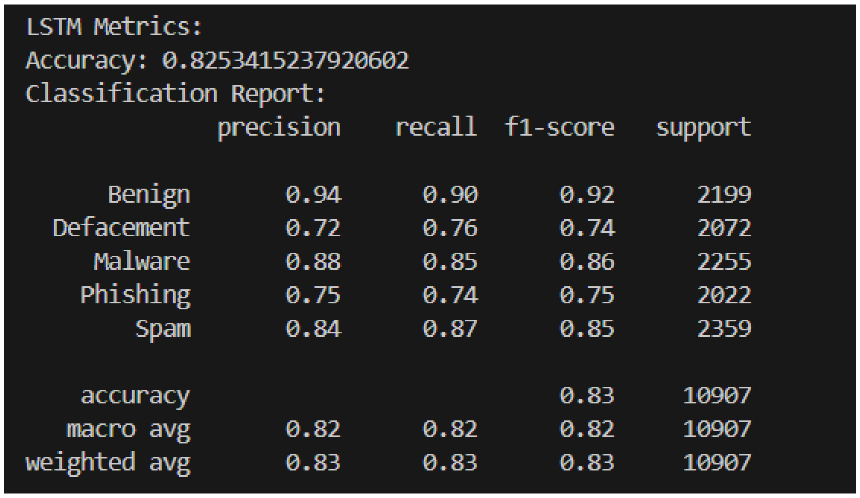


```
CatBoost Metrics:
Accuracy: 0.8922710186119006
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      2199
           1       0.84      0.81      0.82      2072
           2       0.91      0.94      0.93      2255
           3       0.82      0.80      0.81      2022
           4       0.91      0.93      0.92      2359

    accuracy                           0.89     10907
   macro avg       0.89      0.89      0.89     10907
weighted avg       0.89      0.89      0.89     10907
```

Figure 5.2: Catboost Classification Report

5.0.3 LSTM

In our project, Long Short-Term Memory (LSTM) neural networks serve as a powerful tool for sequence modeling and classification tasks. By leveraging their ability to capture long-range dependencies and temporal patterns in sequential data, LSTMs play a crucial role in analyzing and classifying URL features extracted from web data. With an accuracy rate of 83%, our LSTM model demonstrates its effectiveness in accurately categorizing URLs into different classes, such as benign, defacement, malware, phishing, and spam. Through sequential processing of URL feature sequences, LSTM models can effectively discern subtle patterns and characteristics indicative of various cybersecurity threats, thereby enhancing the overall robustness and reliability of our classification system. Additionally, LSTM networks offer the flexibility to handle variable-length input sequences, making them well-suited for processing diverse and dynamic web data encountered in real-world scenarios.

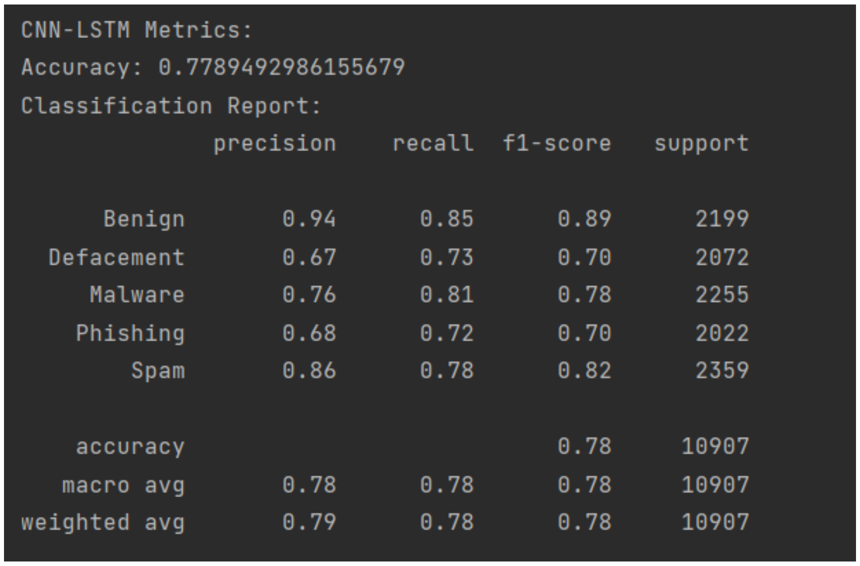


```
LSTM Metrics:
Accuracy: 0.8253415237920602
Classification Report:
              precision    recall  f1-score   support

      Benign       0.94      0.90      0.92      2199
 Defacement       0.72      0.76      0.74      2072
     Malware       0.88      0.85      0.86      2255
    Phishing       0.75      0.74      0.75      2022
        Spam       0.84      0.87      0.85      2359

    accuracy                           0.83     10907
   macro avg       0.82      0.82      0.82     10907
weighted avg       0.83      0.83      0.83     10907
```

Figure 5.3: LSTM Classification Report

5.0.4 CNN-LSTM

In our project, Convolutional Neural Network-Long Short-Term Memory (CNN-LSTM) models serve as a versatile approach for processing and classifying URL features extracted from web data. By combining the strengths of CNNs in spatial feature extraction and LSTMs in sequential modeling, CNN-LSTM architectures excel in capturing both local patterns and long-range dependencies present in URL feature sequences. With an accuracy rate of 78%, our CNN-LSTM model demonstrates its efficacy in accurately categorizing URLs into different classes, including benign, defacement, malware, phishing, and spam. By leveraging the hierarchical representation learning capabilities of CNNs and the temporal modeling capabilities of LSTMs, CNN-LSTM models can effectively discern intricate patterns and subtle characteristics indicative of cybersecurity threats. This hybrid architecture offers enhanced resilience against noise and variability in web data, making it a valuable asset in our classification system for cybersecurity applications.

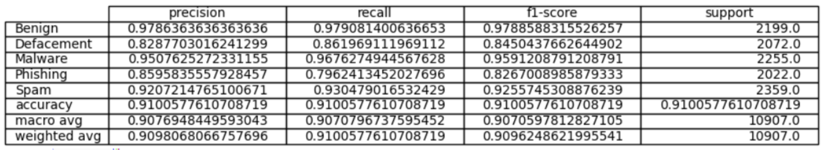


```
CNN-LSTM Metrics:
Accuracy: 0.7789492986155679
Classification Report:
              precision    recall  f1-score   support

      Benign       0.94      0.85      0.89      2199
  Defacement       0.67      0.73      0.70      2072
     Malware       0.76      0.81      0.78      2255
    Phishing       0.68      0.72      0.70      2022
        Spam       0.86      0.78      0.82      2359

    accuracy                           0.78     10907
   macro avg       0.78      0.78      0.78     10907
weighted avg       0.79      0.78      0.78     10907
```

Figure 5.4: CNN-LSTM Classification Report

5.0.5 Random Forest

In our project, the Random Forest algorithm plays a pivotal role in classifying URLs based on their extracted features. With an impressive accuracy rate of 93%, Random Forest demonstrates its robustness and versatility in handling complex web data and discerning patterns indicative of different cybersecurity threats and emerged as the best performing model. Unlike traditional decision trees, Random Forest employs an ensemble approach, where multiple decision trees are trained on different subsets of the data and their outputs are aggregated to make final predictions. This ensemble strategy mitigates overfitting and enhances generalization performance, making Random Forest particularly effective in handling high-dimensional feature spaces and noisy datasets, common characteristics of web data. Moreover, the algorithm's inherent ability to evaluate feature importance provides valuable insights into the underlying mechanisms driving classification decisions, aiding in the interpretation and understanding of the model's behavior. In our project, Random Forest serves as a reliable and high-performing classifier, contributing significantly to the overall effectiveness of our cybersecurity classification system.



	precision	recall	f1-score	support
Benign	0.9786363636363636	0.979081400636653	0.9788588315526257	2199.0
Defacement	0.8287703016241299	0.861969111969112	0.8450437662644902	2072.0
Malware	0.9507625272331155	0.9676274944567628	0.9591208791208791	2255.0
Phishing	0.8595835557928457	0.7962413452027696	0.8267008985879333	2022.0
Spam	0.9207214765100671	0.930479016532429	0.9255745308876239	2359.0
accuracy	0.9100577610708719	0.9100577610708719	0.9100577610708719	0.9100577610708719
macro avg	0.9076948449593043	0.9070796737595452	0.9070597812827105	10907.0
weighted avg	0.9098068066757696	0.9100577610708719	0.9096248621995541	10907.0

Figure 5.5: Random Forest Classification Report

The confusion matrix is a vital tool for evaluating the performance of our classification model t. The confusion matrix provides a detailed breakdown of the model's predictions compared to the actual ground truth labels across different categories, such as benign, defacement, spam, malware, and phishing URLs. It consists of four main components: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Each cell in the matrix represents the count of instances where the model correctly or incorrectly classified a sample. By analyzing the values in the confusion matrix, we can calculate various evaluation metrics such as accuracy, precision, recall, and F1-score, which offer insights into the model's performance in terms of classification accuracy and error rates for each class.

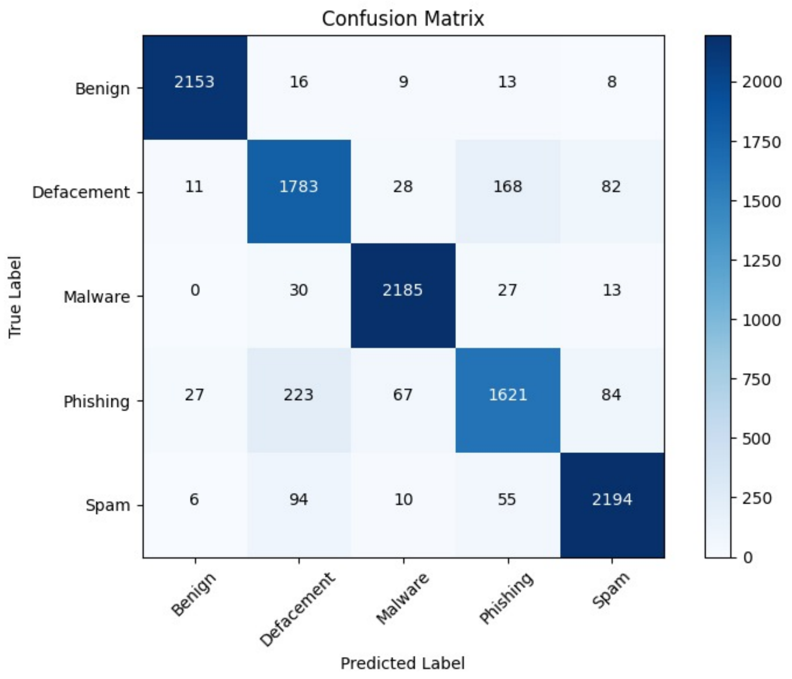


Figure 5.6: Random Forest confusion matrix

The Receiver Operating Characteristic (ROC) curve serves as a critical evaluation metric for assessing the performance of our machine learning and deep learning models in classifying URLs into different categories, such as benign, defacement, spam, malware, and phishing. The ROC curve visually represents the trade-off between the true positive rate (sensitivity) and the false positive rate (1 - specificity) as the classification threshold varies. By plotting the true positive rate against the false positive rate across different threshold values, the ROC curve provides insights into the model's ability to discriminate between the positive and negative classes. A higher area under the ROC curve (AUC) indicates better discriminative performance, with an AUC of 1 representing perfect classification, while an AUC of 0.5 signifies random guessing. In our project, we leverage the ROC curve to compare the performance of different models and select the one with the highest AUC, thereby ensuring optimal classification accuracy and effectiveness in identifying malicious URLs while minimizing false positives.

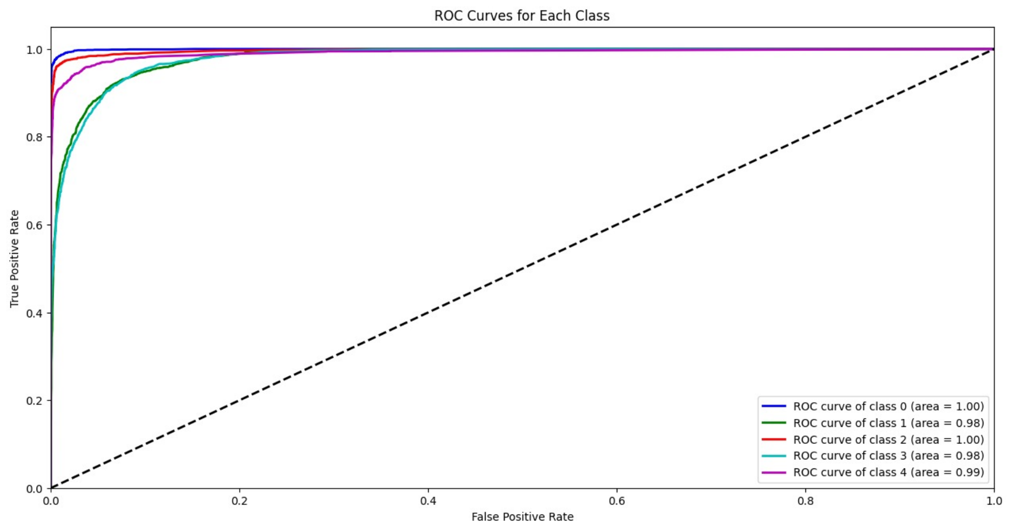


Figure 5.7: Random Forest Receiver Operating Characteristic

Feature engineering is the process of transforming raw data into a format that a machine learning model can understand. In this case, the raw data is a URL, and the features are the different parts of the URL that can be used to classify it. The x-axis of the graph shows the different features, and the y-axis shows the importance of each feature. The importance is a score that indicates how much a particular feature contributes to the model's ability to classify a URL. The higher the score, the more important the feature is. The most important feature

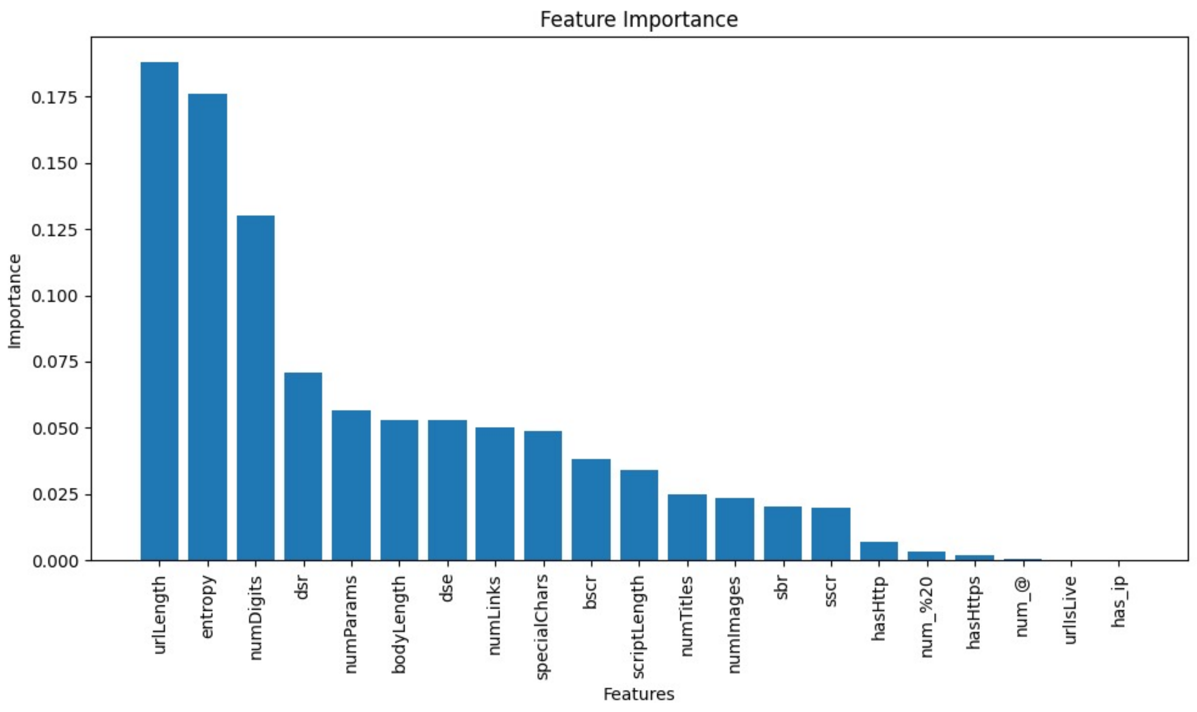


Figure 5.8: Random Forest feature importance

in the graph is urlLength, which refers to the length of the URL. This is followed by entropy, which is a measure of the randomness of the characters in the URL. Other important features include numDigits (the number of digits in the URL), dsr (the ratio of the number of digits to the number of alphanumeric characters), numParams (the number of parameters in the URL), and bodyLength (the length of the body of the text). These features are all important for URL classification because they can help to identify the type of URL. For example, a URL that is long and contains a lot of random characters is more likely to be a spam URL than a URL that is short and contains mostly letters. Similarly, a URL that contains a lot of digits is more likely to be a URL that points to a specific file on a server, such as an image or a PDF file. By understanding the importance of different features, machine learning engineers can improve the accuracy of their URL classification models. For example, they may choose to weight certain features more heavily than others, or they may remove features that are not important for classification.

5.1 Model Comparision Results

Random Forest algorithm often outperforms other models in classification tasks due to several key advantages. One of its notable strengths lies in its robustness to noisy data and outliers. By aggregating predictions from multiple decision trees trained on random subsets of the data, Random Forest effectively reduces the impact of individual noisy data points, resulting in more stable and accurate predictions. Moreover, its ensemble learning approach mitigates overfitting and generalization errors, making it particularly suitable for complex datasets with non-linear relationships between features and the target variable. Additionally, Random Forest provides insights into feature importance, facilitating better understanding of data patterns and aiding in feature selection or engineering. Its ease of use and minimal hyperparameter sensitivity further contribute to its popularity among practitioners. Furthermore, Random Forest's ability to handle class imbalance and generalize well to unseen data enhances its applicability across various domains.

XGBoost and CatBoost:

Limitation: XGBoost and CatBoost are gradient boosting algorithms that rely on sequential decision trees for classification. While they offer excellent performance in many scenarios, they might struggle with non-linear relationships and complex interactions between features, especially when dealing with high-dimensional data.

Comparison: RF, on the other hand, constructs an ensemble of decision trees with random feature subsets, which helps capture non-linear relationships more effectively. This allows RF to perform better when dealing with high-dimensional data or datasets with complex feature interactions.

LSTM (Long Short-Term Memory):

Limitation: LSTM is a type of recurrent neural network (RNN) specifically designed for sequence data. However, LSTM models have limitations in handling sequences with long dependencies or complex temporal patterns. Additionally, LSTMs typically require fixed-length input sequences, limiting their applicability to scenarios where the sequence length is predefined.

Comparison: RF excels in scenarios where the relationship between features is not sequential or temporal. Unlike LSTM, RF does not require fixed-length input sequences and can handle variable-length feature vectors efficiently. This makes RF a better choice for analyzing non-sequential data or datasets with complex feature interactions that do not exhibit temporal dependencies.

CNN-LSTM (Convolutional Neural Network - Long Short-Term Memory):

Limitation: CNN-LSTM combines the strengths of CNNs in extracting spatial features and LSTMs in capturing temporal dependencies. However, CNN-LSTM architectures may suffer from limited interpretability and scalability issues, especially when dealing with high-dimensional data or large-scale datasets.

Comparison: RF offers superior interpretability and scalability compared to CNN-LSTM. While CNN-LSTM may perform well in scenarios where spatial and temporal features are crucial, RF's ability to handle high-dimensional data and provide feature importance rankings makes it more suitable for a wide range of applications, particularly those requiring explainable and scalable solutions.

Algorithm	**Accuracy**	**Precision**	**Recall**	**F1-score**
XgBoost	0.913	0.912	0.912	0.910
Catboost	0.892	0.888	0.890	0.890
Random Forest	**0.918**	**0.916**	**0.918**	**0.913**
LSTM	0.825	0.826	0.824	0.824
CNN-LSTM	0.779	0.782	0.778	0.778

Table 5.1: Model Comparison Table

In summary, Random Forest's combination of robustness, flexibility, and ease of implementation makes it a go-to choice for classification tasks, often yielding superior accuracy compared to alternative algorithms. Its ability to handle various data characteristics, including noise, non-linearity, and high dimensionality, along with its interpretability and generalization capabilities, solidify its position as a versatile and reliable model for this project.

5.2 Final Application

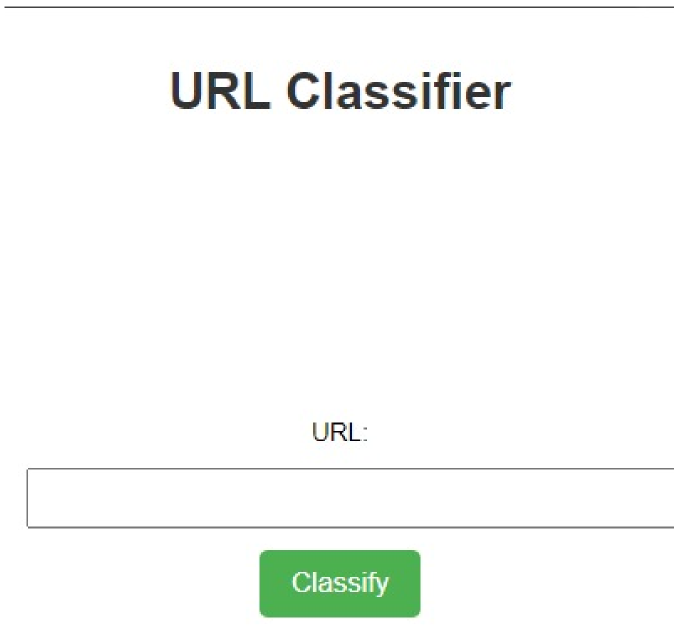


Figure 5.9: Browser Plugin user interface

This figure depicts a plugin interface prompting users to input the URL of a potentially malicious website for inspection. After entering the URL and clicking on the ’Classify’ button, the plugin initiates a request to the backend for classification. Upon completion of the classification process, the user is presented with another text box, allowing them to pose queries regarding the model or the URL.

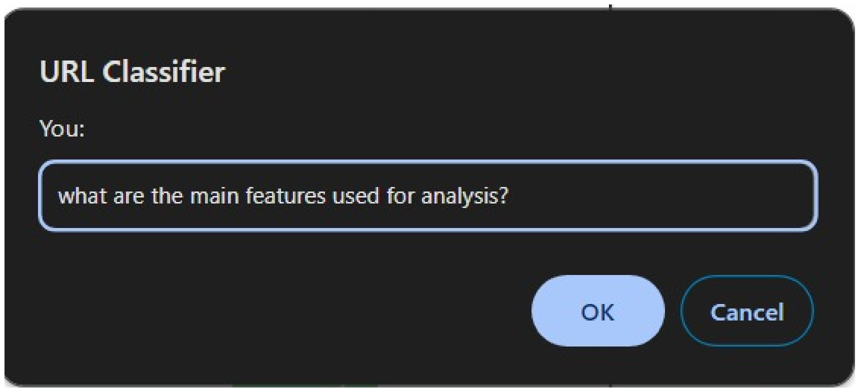
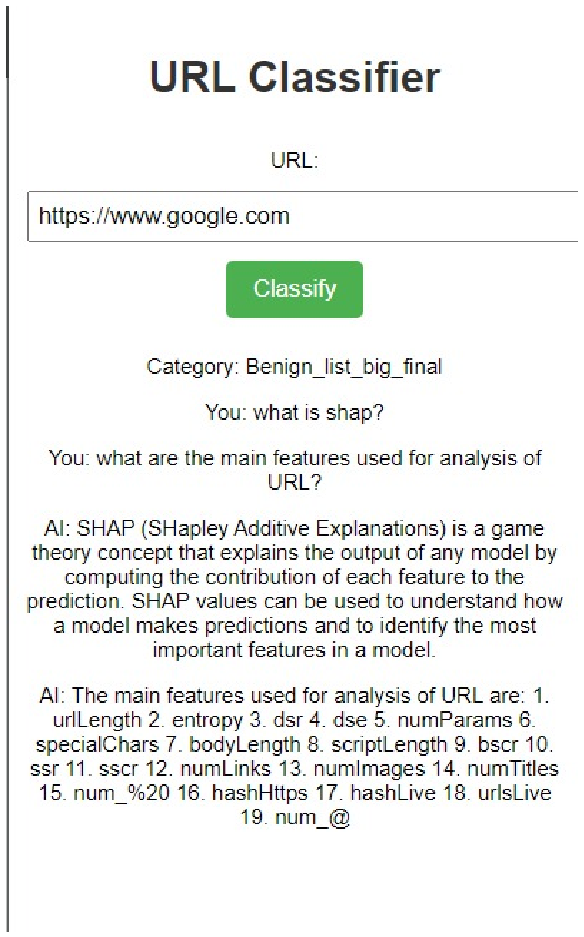


Figure 5.10: LLM user query box

A user-friendly pop-up interface emerges, serving as a conduit for users to interact with the Gemini model operating in the backend. This interface offers a seamless platform for users to pose inquiries to the Gemini model, leveraging its robust capabilities for insightful responses. This interface facilitates users to gain deeper insights into the classification process and the underlying features of the analyzed URLs.

A sophisticated JavaScript code orchestrates a seamless user experience on the frontend interface. Upon the user clicking the "classify" button, the URL input is extracted. This URL is then dispatched to the backend server through a POST request to a designated endpoint. The backend meticulously processes the URL, determines its category, and swiftly returns the result, which is dynamically displayed on the webpage. In the event of obtaining a category, the code seamlessly transitions to initiate interaction with an AI model.This function actively prompts the user for queries, subsequently sending them to the backend server via POST requests to the endpoint. Real-time AI-generated responses are elegantly displayed inside the plugin, enriching user engagement and interaction.


URL Classifier

URL:

https://www.google.com

Classify

Category: Benign_list_big_final

You: what is shap?

You: what are the main features used for analysis of URL?

AI: SHAP (SHapley Additive Explanations) is a game theory concept that explains the output of any model by computing the contribution of each feature to the prediction. SHAP values can be used to understand how a model makes predictions and to identify the most important features in a model.

AI: The main features used for analysis of URL are: 1. urlLength 2. entropy 3. dsr 4. dse 5. numParams 6. specialChars 7. bodyLength 8. scriptLength 9. bscr 10. ssr 11. sscr 12. numLinks 13. numImages 14. numTitles 15. num_%20 16. hashHttps 17. hashLive 18. urlsLive 19. num_@

Figure 5.11: Browser Plugin user interface

Illustrated here is the culminating segment of the plugin, wherein the input URL undergoes precise classification into one of five distinct categories: benign, defacement, malware, spam, or phishing. Furthermore, within the plugin interface, users' queries are seamlessly paired with the corresponding model responses, elegantly encapsulating the iterative interaction between users and the model.

Chapter 6

CONCLUSION

In our endeavor to bolster cybersecurity measures, our project embarked on a quest to revolutionize URL classification and threat detection. We commenced by meticulously assembling a comprehensive dataset, encompassing five distinct classes of URLs: benign, malware, spam, phishing, and defacement. This dataset served as the bedrock for our analysis, enabling us to extract a rich array of 34 informative features crucial for effective classification. With this robust dataset, we delved into a thorough exploration of machine learning algorithms. Our experimentation encompassed a diverse spectrum, ranging from traditional models like Random Forest, XGBoost, and CatBoost to more intricate architectures such as LSTM and CNN-LSTM. Through rigorous analysis and evaluation, Random Forest emerged as the standout performer, demonstrating unparalleled efficacy in URL classification tasks. This discovery underscored the importance of ensemble methods and decision tree algorithms in the realm of cybersecurity. The integration of Explainable AI techniques, particularly SHAP graphs, offered invaluable insights into the inner workings of our models, fostering transparency and instilling confidence in their reliability. Additionally, the incorporation of Google's Gemini model enriched our system, furnishing users with an intuitive and interactive interface for real-time query addressing. Recognizing the need for seamless integration into existing cybersecurity infrastructures, we meticulously designed our system as a plugin model. This approach ensured effortless adoption and deployment, offering cybersecurity professionals a practical and user-friendly solution for URL classification and threat detection. In conclusion, our project represents a significant milestone in the ongoing battle to safeguard digital ecosystems against emerging threats. By embracing a multifaceted approach that combines advanced machine learning techniques, explainable AI methodologies, and interactive interfaces, we empower users with the tools and insights necessary to navigate the complex landscape of online threats with confidence and resilience.

REFERENCES

[1] Joshi, A., Lloyd, L., Westin, P. and Seethapathy, S., 2019. Using lexical features for malicious URL detection–a machine learning approach. arXiv preprint arXiv:1910.06277

[2] S. He, J. Xin, H. Peng and E. Zhang, "Research on Malicious URL Detection Based on Feature Contribution Tendency," 2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), Chengdu, China, 2021, doi: 10.1109/ICCCBDA51879.2021.9442606

[3] R. Raj and S. Singh Kang, "Spam and Non-Spam URL Detection using Machine Learning Approach," 2022 3rd International Conference for Emerging Technology (INCET), Belgaum, India, 2022, pp. 1-6, doi: 10.1109/INCET54531.2022.9825197.

[4] H. BOUIJIJ and A. BERQIA, "Machine Learning Algorithms Evaluation for Phishing URLs Classification," 2021 4th International Symposium on Advanced Electrical and Communication Technologies (ISAECT), Alkhobar, Saudi Arabia, 2021, pp. 01-05, doi: 10.1109/ISAECT53699.2021.9668489.

[5] U. S. D. R, A. Patil and Mohana, "Malicious URL Detection and Classification Analysis using Machine Learning Models," 2023 International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT), Bengaluru, India, 2023, pp. 470-476, doi: 10.1109/IDCIoT56793.2023.10053422.

[6] B. Gogoi, T. Ahmed and A. Dutta, "A Hybrid approach combining blocklists, machine learning and deep learning for detection of malicious URLs," 2022 IEEE India Council International Subsections Conference (INDISCON), Bhubaneswar, India, 2022, pp. 1-6, doi: 10.1109/INDISCON54605.2022.9862909.

[7] Q. Jia et al., "Phishing URL recognition based on ON-LSTM attention mechanism and XGBoost model," 2023 5th International Conference on Electronics and Communication, Network and Computer Technology (ECNCT), Guangzhou, China, 2023, pp. 159-163, doi: 10.1109/ECNCT59757.2023.10280927.

[8] J. K. S and A. B, "Phishing URL detection by leveraging RoBERTa for feature extraction and LSTM for classification," 2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), Trichy, India, 2023, pp. 972-977, doi: 10.1109/ICAISS58487.2023.10250684.

[9] F. Ren, Z. Jiang and J. Liu, "A Bi-Directional LSTM Model with Attention for Malicious URL Detection," 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chengdu, China, 2019, pp. 300-305, doi: 10.1109/IAEAC47372.2019.8997947.

[10] Aslam N Khan, I.U Mirza S, Alowayed A Anis, F.M, Aljuaid R.M, Baageel, R, "Interpretable Machine Learning Models for Malicious Domains Detection Using Explainable Artificial Intelligence (XAI)". Sustainability 2022