

SECURE AND SCALABLE MALICIOUS URL DETECTION USING MACHINE LEARNING AND SERVERLESS COMPUTING



By

Sikandar Shaheen

(Registration No: 00000431932)

A thesis submitted to the National University of Sciences and Technology, Islamabad,

in partial fulfillment of the requirements for the degree of

MSIS-21

Supervisor: Dr. Shahzaib Tahir

Military College of Signals (MCS)


National University of Sciences and Technology (NUST)

Islamabad, Pakistan

(2024)

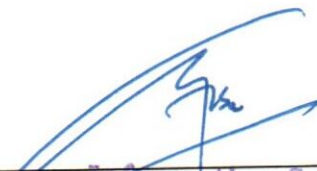
THESIS ACCEPTANCE CERTIFICATE


Certified that final copy of MS Thesis written by **Sikandar Shaheen** Registration No. **00000431932**, of **Military College of Signals** has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations/MS Policy, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members and local evaluators of the scholar have also been incorporated in the said thesis.

Signature: _____ 

Name of Supervisor **Dr. Shahzaib Tahir, PhD**

Date: 11/10/24

Signature (HOD): _____  **HoD**
Information Security
Date: 11/10/24 **Military College of Sigs**

Signature (Dean/Principal) _____ 
Date: 11/10/24 **Brig**
Dean, MCS (NUST)
(Asif Masood, Phd)

NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY
MASTER THESIS WORK


We hereby recommend that the dissertation prepared under our supervision by **Sikandar Shaheen, MSIS-21 Course** Regn No **00000431932** Titled: "**Secure and Scalable Malicious URL Detection using Machine Learning and Serverless Computing**" be accepted in partial fulfillment of the requirements for the award of **MS Information Security** degree.

Examination Committee Members

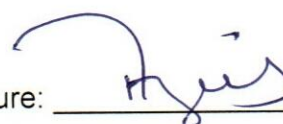
1. Name: **Dr. Sadiqa Arshad**

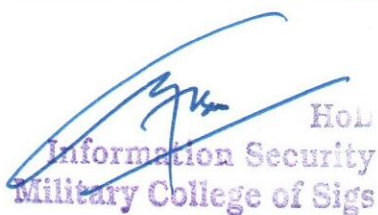
Signature: 

2. Name: **Engr Bilal Ahmed**

Signature: 

Supervisor's Name: **Dr. Shahzaib Tahir, PhD**

Signature: 


HOD
Information Security
Military College of Sigs


Head of Department

Date: 11/10/24

11/10/24
Date

COUNTERSIGNED

Date: 11/10/24


Brig
Dean, MCS (NUST)
(Asif Masood, Phd)
Dean

CERTIFICATE OF APPROVAL


This is to certify that the research work presented in this thesis, entitled "Secure and Scalable Malicious URL Detection using Machine Learning and Serverless Computing" was conducted by Sikandar Shaheen under the supervision of Dr. Shahzaib Tahir. No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the Military College of Signals, National University of Science & Technology Information Security Department in partial fulfillment of the requirements for the degree of Master of Science in Field of Information Security Department of information security National University of Sciences and Technology, Islamabad.

Student Name: Sikandar Shaheen

Signature: 

Examination Committee:


a) External Examiner 1: Dr. Sadiqa Arshad (MCS)

(MCS) Signature: 


b) External Examiner 2: Maj Bilal Ahmed (MCS)

(MCS) Signature: 

Name of Supervisor: Dr. Shahzaib Tahir

Signature: 

Name of Dean/HOD: Dr Muhammad Faisal Amjad

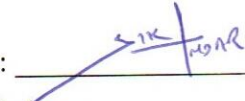
Signature: 
HOD
Information Security
Military College of Sigs
11/10/24

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled "Secure and Scalable Malicious URL Detection using Machine Learning and Serverless Computing" is solely my research work with no significant contribution from any other person. Small contribution/ help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero-tolerance policy of the HEC and National University of Sciences and Technology (NUST), Islamabad towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS degree, the University reserves the rights to withdraw/ revoke my MS degree and that HEC and NUST, Islamabad has the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized thesis.

Student Signature: 
Name: Sikandar Shaheen
Date: 11/10/24

AUTHOR'S DECLARATION

I Sikandar Shaheen hereby state that my MS thesis titled "Secure and Scalable Malicious URL Detection using Machine Learning and Serverless Computing" is my own work and has not been submitted previously by me for taking any degree from National University of Sciences and Technology, Islamabad or anywhere else in the country/ world. At any time if my statement is found to be incorrect even after I graduate, the university has the right to withdraw my MS degree.

Student Signature: _____

Name: Sikandar Shaheen

Date: _____

11/10/24

Dedication

I sincerely dedicate my research work to Allah Mighty, the source of wisdom and knowledge, to which my inspiration has never been exhausted, and this journey could not have been so spanned. Moreover, I dedicate this work to my devoted parents, enlightened teachers, and kind family, whose unlimited love, care, and support has become the incentive for me to continuously advance to discovering outstanding opportunities. On the one hand, I am highly obliged to my parents for their hard work, discipline, and dedication to achieve high results. On the other hand, their support and assistance regarding this research was also paramount and immeasurable. Hence, my thesis presents a reflection of my cumulative knowledge and wisdom I owe to my dear parents.

ACKNOWLEDGEMENTS

First, I want to say thank you to all people who supported me, considering this research work. My deep gratitude to the people that were assisting me at any part of it and guiding my road from the first to last point.

I would like to use this opportunity to say thank you to my supervisor, Dr. Shahzaib Tahir for helping me in doing research. His supervision and direction got me through all the difficulties, rooted in my research endeavors. The motivation, the materials, education system, and gained experience were solved with extremely clever thoughts. I am grateful to Dr. Shahzaib for affecting my academic growth, as well as in my research.

I would like to express my heartfelt gratitude to the esteemed faculty at MCS, NUST for fostering an intellectually stimulating academic environment. Their expertise, teaching skills, thought-provoking lectures, and engaging discussions have greatly enhanced my knowledge and profoundly shaped my perspective on the latest technologies.

Contents

LIST OF TABLES	VI
LIST OF FIGURES	VII
LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS	VIII
ABSTRACT	IX
1 Introduction	1
1.1 Background	1
1.2 Problem Statement	3
1.3 Research Questions	4
1.3.1 RQ1: To what extent do machine learning approaches assist in recognizing malicious URLs utilizing characterizing URL strings and extracting data from the web?	4
1.3.2 RQ2: What benefits can be identified with the implementation of the malicious URL detection system that is operated based on serverless computing technologies regarding its scalability, efficiency and security of the operation?	4
1.3.3 RQ3: How can the performance of the proposed system be de- fined against the application of blacklistings, heuristic analyses, and signatures or combination thereof?	5
1.3.4 RQ4: What issues are encountered when using machine learn- ing in combination with serverless computing with the purpose of creating a cyber defense system?	5
1.4 Research Objectives	5
1.4.1 Objective 1: Develop a machine learning-based approach for feature extraction from URL strings and content analysis of website data to detect malicious URLs.	5

1.4.2	Objective 2: Implement the malicious URL detection system on a serverless computing architecture to enhance scalability, efficiency, and security.	6
1.4.3	Objective 3: Evaluate the performance of the proposed solution in terms of detection accuracy, false positive/negative rates, scalability, and resource efficiency.	6
1.4.4	Objective 4: Assess the effectiveness of integrating machine learning models with serverless computing for malicious URL detection compared to traditional detection methods.	7
1.5	Thesis Structure	7
1.5.1	Chapter 1: Introduction	7
1.5.2	Chapter 2: Literature Review	7
1.5.3	Chapter 3: Methodology	7
1.5.4	Chapter 4: Results and Analysis	8
1.5.5	Chapter 5: Discussion	8
1.5.6	Chapter 6: Conclusion	8
2	Literature Review	9
2.1	Background	9
2.2	Phishing and Malicious URLs	9
2.2.1	Evolution of Phishing Attacks	9
2.2.2	Traditional Detection Methods	10
2.2.3	Advanced Detection Techniques	11
2.3	Machine Learning and Cybersecurity	13
2.3.1	Insight into Machine Learning Frameworks	13
2.3.2	Feature Engineering for URL Identification	14
2.3.3	Challenges in Applying Machine Learning to Cybersecurity	15
2.4	Serverless Computing in Cybersecurity	16
2.4.1	Overview of Serverless Computing	16
2.4.2	Benefits of Serverless Computing for Cybersecurity	17
2.4.3	Challenges and Limitations of Serverless Computing	17
2.5	Integrating Machine Learning with Serverless Computing	18
2.5.1	Architectural Considerations	18
2.5.2	Case Studies and Applications	18
2.6	Summary	20

3	Methodology	22
3.1	Introduction	22
3.2	Research Design	23
3.2.1	Overview of the Research Process	23
3.2.2	Data Collection	25
3.2.2.1	Malicious URLs	25
3.2.2.2	Benign URLs	25
3.2.3	Data Cleaning and Preprocessing	26
3.2.3.1	Live URL Validation	26
3.2.3.2	Duplicate Removal	26
3.2.3.3	Balancing the Dataset	27
3.3	Feature Engineering	27
3.3.1	Lexical Features	28
3.3.2	Content-Based Features	30
3.4	Machine Learning Model Development	31
3.4.1	AdaBoost	32
3.4.2	XGBoost	33
3.4.3	Random Forest	33
3.4.4	Extra Trees	34
3.4.5	K-Nearest Neighbors (KNN)	34
3.4.6	Stochastic Gradient Descent (SGD) Classifier	35
3.4.7	Naive Bayes	36
3.5	System Deployment	37
3.5.1	Local System Deployment	38
3.5.2	Google Cloud Functions for Feature Extraction	39
3.5.3	System Components in Google Cloud Functions:	39
3.5.4	API Gateway and Routing	39
3.5.5	Data Storage and Integration	40
3.5.6	Integration with Local System	40
3.5.7	Hardware Specifications	40
3.5.8	Key Benefits of Hybrid Deployment	40
3.6	Cross-Validation	41
3.6.1	Cross-Validation Process	42
3.6.2	Performance Metrics Calculated	43

4	Results and Analysis	44
4.1	Model Performance	44
4.1.1	Detailed Observations	44
4.2	Feature Importance	48
4.2.1	Key Findings	49
4.3	Scalability and Security Evaluation	49
4.3.1	Benefits of Using Google Cloud Functions	50
4.3.2	Final Model Selection	51
4.3.2.1	Why XGBoost Was Chosen	52
5	Conclusion	53
5.1	Implications of Findings	53
5.2	Limitations	54
5.3	Future Work	54
5.4	Summary of Key Contributions	55
5.5	Final Thoughts	56

List of Tables

2.1	Comparison of Traditional Detection Techniques	11
2.2	Comparison and Analysis of ML Algorithms	13
2.3	Literature Review Summary	19
3.1	Comparison of Machine Learning Algorithms	36
4.1	Performance Assessment of Machine Learning Techniques	44
4.2	Local vs Google Cloud Function Processing	51

List of Figures

2.1	Machine Learning-Based URL Detection	12
2.2	Serverless Computing in General	16
3.1	General Architecture	23
3.2	Google Cloud Function and Storage	24
3.3	Google Cloud Function and Storage	28
3.4	Machine Learning Model	32
3.5	GUI Interface of Web based Flask Applications	38
3.6	Cross-Validation Steps	41
3.7	Cross-Validation Accuracy Comparison	43
4.1	XGBoost – Confusion Matrix	45
4.2	Random Forest – Confusion Matrix	45
4.3	Extra Trees – Confusion Matrix	46
4.4	AdaBoost – Confusion Matrix	46
4.5	Decision Tree – Confusion Matrix	47
4.6	KNN – Confusion Matrix	47
4.7	Naive Bayes – Confusion Matrix	48
4.8	SGD – Confusion Matrix	48
4.9	Features Importance Score for XGBoost	49
4.10	Average Feature Extraction Time	50
4.11	Classifiers Execution Time Comparison	51

ABBREVIATIONS AND ACRONYMS

ML : Machine Learning
URL : Uniform Resource Locator
CNN : Convolutional Neural Networks
RNN : Recurrent Neural Networks
SVM : Support Vector Machine
DL : Deep Learning
XGBoost : eXtreme Gradient Boosting
GRU : Gated Recurrent Unit
IoT : Internet of Things

Abstract

Malicious URLs have become major threat vectors over the Internet, with attackers using URLs to launch attacks such as phishing campaigns, malware distribution, or even data exfiltration. Naturally, since malicious URLs are modeled after real ones, they can be difficult for users to spot and identify as frauds. A single attack can target an organization with thousands of malicious URLs, costing in data loss, financial losses and reputation damages. Even worse, cyber criminals are fast at revising their tactics and this makes the identification a frustration, which requires effective countermeasures. This thesis presents a novel hybrid model of machine learning and serverless computing to tackle the challenge of detecting malicious URLs. In this research, I am using a well-balanced dataset of 48,000 consistent URLs, from reputable sources such as PhishTank and VirusTotal. Using such a varied dataset achieves the purpose of training on benign as well as malicious URLs, thereby assisting the model to learn better and generalize well across various cyber threats. Through features extracting process, I have selected 54 different features (25 from the URL strings and 29 from the content of respective web pages) for identification of malicious URLs. Multiple machine learning model were tested and evaluated including Decision tree, Random Forest, AdaBoost. In the end, XGBoost emerged as the standout performer, achieving an impressive accuracy of 98.14%. The testing showed the potential of hybrid model in a high detection rate regarding malicious URLs and improved amount in processing time that was saved significantly by the serverless architecture. Serverless computing also provided sandbox like environment for securely extraction of features from web-page content. This research exhibits the efficiency of my hybrid model in precisely identifying malicious URLs and showcases the potential of merging machine learning with serverless computing to bolster our defenses against evolving cyber threats.

Keywords: Malicious URL Detection, Machine learning, Serverless computing, Cybersecurity, Feature Extraction, Scalability, Real-time Analysis

Chapter 1

Introduction

1.1 Background

The rapid rise and subsequent effect of the internet and digital services on nearly all aspects of our lives from living to working and communicating is quite evident. These advancements opened numerous opportunities for various forms of innovation and efficiency. At the same time, tectonic changes naturally engender numerous complications in terms of internet security. One of the most highlighted of these was the rise of malicious URLs. Essentially, a malicious URL is a fraudulent website address constructed in such a way as to encourage user visits to a harmful website where malware can be downloaded, phishing can be successfully carried out, or other forms of harmful influence can be propagated. The methods of creating such URLs have become quite sophisticated and intricate, allowing the criminals to develop elaborate websites that cannot be easily recognized as harmful.

On one hand, these changes made the methods of URL identification difficult – they no longer can be deemed as obviously bad and avoided. On the other hand, these changes made the existing layers of internet security measures, such as blacklisting URLs with negative reputations and the employment of signature-based URL detection, increasingly irrelevant. These changes suggest the necessity of unveiling increasingly new forms of URL detection algorithms immune to the development of intricate new forms of malware. In this context, the evolution of URL detection methods towards machine learning solutions is hardly surprising. Unlike traditional rules-based systems, these solutions are predicated on massive volumes of recordings and the ability of the machine learning algorithm to recognize vastly more intricate patterns of what is initially deemed dangerous, especially as the nature of cyber-activity becomes more and more sophisticated.

Furthermore, machine learning-based systems provide a level of flexibility and precision that can be upgraded as new data is added to the system. This helps them get better and better at dealing with never-seen-before threats. But with internet usage booming out of proportion and online users and traffic growing at an exponential rate, then that is not the only challenge modern detection systems have to face. With millions – or billions – of URLs accessed every single day only to grow to hundreds of billions more, detection systems have to be not only precise but also extremely scalable in order to accommodate the volume and manage data efficiently. This means having a sturdy infrastructure that allows for scaling without affecting performance, latency, or precision. As such, ensuring that the ML model remains effective at classifying URLs despite the insane growth of data is key to any real-world deployed model.

Serverless computing is a relatively recent technological advancement that has become a key driver in addressing the scalability and resource issues that modern systems face. It is an architecture in which a cloud provider pushes all server and infrastructure management to cloud service providers. This essentially allows a developer to focus solely on their application’s core functionality and optimize them for better efficiency. In a serverless infrastructure, developers are not required to manage and configure servers – they simply have to deploy the code, and everything else, from server allocation to load balancing and automatically scaling, will be managed and adjusted according to the traffic demand by the cloud provider.

Apply this, and the scaling process is greatly simplified because there is no need to deal with the intricacies of managing hardware and even lesser so with the complexities of various infrastructures. And this is especially true in situations where traffic fluctuates. Serverless applications provide an event-driven model that is elastic and on-demand; they automatically adjust to allocate and deallocate resources required by the application at any given time. This means applications can scale in real-time automatically without having to worry about over provisioning and wasting money or under-provisioning and losing time.

Serverless computing does not simply offer flexible scalability but serves as a security enhancement as well. For example, when analyzing the nature of URLs to identify potentially dangerous ones, a serverless function works in the manner of a sandbox. Each of the functions executed in a serverless environment works in a separate environment. As a result, if a sample URL was analyzed and proved presumable dangerous, it directly affects the environment in which the specific function was executed. In the present instance, the concern about failing to determine whether or not a URL is harmful will not result in any strain between other functions or affect the rest of the system. Thus, the serverless architecture “protects misbehaving / buggy / malicious functions from themselves, or the system” through the protection of underlying

resources. It should be noted that such an approach results in a win-win situation. On the one hand, serverless architecture is impeccable in providing the necessary flexibility to adjust the scalability of the system in case of varying workloads. On the other hand, protection of resources through isolation allows systems to remain safe, unaffected by malicious intent and secure in the context of security malware.

The other essential argument in favor of serverless architecture, as the technology to be used for the detection of URL-based malware, is its low cost. Specifically, in the case of serverless architecture, the pay-as-you-go model implies that a system analysis of URLs to detect malicious intent only costs the company as much as the resources that the specified analysis materials consume during their execution. In other words, they are not required to pay for the servers that sit idle, readying to implement large machine learning algorithms and perform real-time data analysis. Thus, since the serverless architecture model genuinely implies cost-effectiveness, it can be safely assumed that the implementation of machine learning as the basis for cybersecurity systems will help achieve a new level of anti-malware protection.

1.2 Problem Statement

Cyber threat landscape is rapidly changing, especially in terms of malicious URLs. Thus, conventional detection approaches are no longer effective due to the dynamic nature of URL threats. As many malicious URL developers continuously develop new URLs, there is no fixed detection rule to use the URL-based rule. Additionally, with billions of URLs online, URL detection becomes a task that requires accurate results at vast data. On the one hand, machine learning systems are designed to analyze patterns and exploit unusual subtleties that could not be detected by conventional systems. Models are also modified in parallel with new even defined URL threats. However, such implementations also have downsides. For example, the training process for machine learning models is resource intensive. By integrating machine learning algorithms with large datasets, the system faces a huge challenge in maintaining a secure and accurate infrastructure. However, serverless computing appears as an opportunity to solve the challenges that can be generated because serverless applications do not manage infrastructures. Personal and secure platforms that also include models can be created using simple cloud services. However, the combination of serverless computing and machine learning is a new research area. There is a limited number of studies in which serverless architecture and machine learning are used together in cyber security. Thus, among these studies, especially URL threat detection is very few. Overall, within these limitations, there is currently no study focusing on the security and scalability of malicious URL detecting machine learning models with serverless computing.

This research highlights the primary research question which investigates whether it is feasible to create a cost- and energy-efficient solution that can detect malicious URLs using the serverless computing-based machine learning model. Overall, the primary aims of the research are to create an implementation of a machine learning model and, at the same time, URL feature extraction using serverless computing applications. The research has significance because there are no studies in the literature that examine the feasibility and create a model. The created implementation will evaluate not only calculated storage and memory practices but also the robustness and scalability properties of different serverless platforms. The environment will also be designed with completely secure and clean platforms. As the first step, related literatures and current cases in the field were searched. Later, for system design and implementation, Python, Google Cloud Function and Google Cloud Storage were chosen. To summarize, it is possible to create secure and scalable URLs with machine learning training and machine learning model applications using serverless computing applications. At the end of the research, an architecture of machine learning models that can work with serverless applications will be created.

1.3 Research Questions

This research seeks to answer the following key questions:

1.3.1 RQ1: To what extent do machine learning approaches assist in recognizing malicious URLs utilizing characterizing URL strings and extracting data from the web?

The primary target of the proposed research is to estimate the efficacy of numerous machine learning models and feature extraction methods regarding the precision of recognizing malicious URLs.

1.3.2 RQ2: What benefits can be identified with the implementation of the malicious URL detection system that is operated based on serverless computing technologies regarding its scalability, efficiency and security of the operation?

This question is far more related to the research design due to the high probability of identifying such benefits as high compensational patterns, energy efficiency of computations, scalability, accessibility of operations, and cost-effectiveness of the analyses.

Thus, the implementation of such algorithms for the detection of malicious content within URL strings that may cause damage to the statistical performance of serverless systems requires a profound investigation of wits advantages.

1.3.3 RQ3: How can the performance of the proposed system be defined against the application of blacklistings, heuristic analyses, and signatures or combination thereof?

This research question will be associated with the comparative analysis between the proposed system and its application and such mainstream methods of detection like blacklisting, the combination of decision-making, heuristic methods, and signature-based technologies.

1.3.4 RQ4: What issues are encountered when using machine learning in combination with serverless computing with the purpose of creating a cyber defense system?

Finally, it is possible to state that the following research will help in gaining a notion regarding IT using, as well as the possible issues associated primarily with data management, model deployment, and safeguarding the models.

1.4 Research Objectives

The primary objectives of this research are as follows:

1.4.1 Objective 1: Develop a machine learning-based approach for feature extraction from URL strings and content analysis of website data to detect malicious URLs.

This objective concerns developing, designing, and implementing feature extraction techniques that are capable of precisely distinguishing between benign and malicious URLs. To recognize the latter, the system has to capture many types of data. The first feature extraction step is focused on obtaining features from the URL strings: those include the length of URL, the number of ‘special’ characters, the structure of domain name and domain level, or the presence of suspicious keywords. Such features – also called lexical – are especially important as the malicious intentions of the URL owner are typically masked many ways, such as through URL shortening or domain spoofing.

The content-based features, in turn, are features that can be extracted only from the website which the URL refers to. Such characteristics can include the structure of a webpage – such as the presence of forms, iframes, or scripts – the metadata, or the content itself. In the latter case, one might want to implement searching for malware specific writing style or several types of hyperlinks to suspicious websites or supporting illegal actions. The vast amount of such lexical and content-based features will allow the act to use modern learning algorithms and notice even the most subtle signs of malicious behavior that are usually beyond the reach of classical malware detection algorithms.

1.4.2 Objective 2: Implement the malicious URL detection system on a serverless computing architecture to enhance scalability, efficiency, and security.

This objective focuses on activating the machine learning – based malicious URL detection system on a serverless computing platform. Serverless computing offers the ability to implement an elastic infrastructure that reacts and scales itself in correlation with a sporadic URL traffic. The serverless deployment automatically scales to the number of URLs that need analysis with no server volume limit or applause from the administrator. This ensures that the system can handle large datasets and cannot fail to cover the real – time URL traffic. Moreover, the sand boxed execution of workloads on servers can significantly improve security. This type of security offers a clean space between the code and the operating system, releasing the possibility of cross – infections. Therefore, since the URLs can be harmful on this system, the sand boxed environment isolates the threat from the environment in which the actual work is performed.

1.4.3 Objective 3: Evaluate the performance of the proposed solution in terms of detection accuracy, false positive/negative rates, scalability, and resource efficiency.

The purpose of this objective is to perform a comprehensive evaluation of the performance of the detection system based on several crucial metrics. The first and most crucial factor that will be assessed is the rate of the detection system in terms of detecting the URL as either malicious or benign.

Second, the evaluation will also test the scalability of the system in terms of handling an extensive amount of URLs and ensure that the changes in the scope of analyzed pages does not influence the performance, speed, or rate of detection.

1.4.4 Objective 4: Assess the effectiveness of integrating machine learning models with serverless computing for malicious URL detection compared to traditional detection methods.

This study will highlight the advantages of fusing machine learning with serverless computing for real-time handling, moderately priced development capacity and superior detection accuracy, while also revealing areas that could be further improved. The target of this system is to prove that not only can the proposed method of detection exceed traditional methods in effectiveness, but it delivers a reliable and expandable one which can cope with the steady growth of hacking threats. By tackling these problems, the research aims to establish a more secure and effective framework for practical field tests variable environment abuses, fraudulent URLs.

1.5 Thesis Structure

The thesis is structured as follows:

1.5.1 Chapter 1: Introduction

Offers a comprehensive overview of the research, describing its background, stating the problem clearly, and defining key objectives and research questions. This chapter also explains thesis construction principles guiding readers through subsequent sections.

1.5.2 Chapter 2: Literature Review

Surveys current literature about how malicious URL detection methods machine learning technology and serverless computer systems are deployed in sequence. The researchers hope that their work will fill in some critical gaps in current knowledge, building the foundation for their Shudder to think.

1.5.3 Chapter 3: Methodology

Details how the research is carried out, including study design, data collection methods and the procedures for feature extraction. In addition, this chapter explains how machine learning models were deployed on a serverless platform and describes its technical implementation step by step.

1.5.4 Chapter 4: Results and Analysis

Reports all of the experimental results, this part presents the performance of different machine learning models in detecting malicious URLs. In this chapter these results are compared with traditional methods, pointing out key performance indicators such as accuracy, scalability and efficiency.

1.5.5 Chapter 5: Discussion

Within a broader cybersecurity context, this part interprets the findings of the study and discusses their implications for detecting changing threats. This chapter looks back on the contributions to the field in light of what it has found, then it per speculates improvements possible from here and identifies parts of study which need further research effort or practical application.

1.5.6 Chapter 6: Conclusion

In conclusion, the chapter provides the final evaluation based on the key findings, concluding the system's effectiveness in addressing the identified issues. It incorporates the issues identified and the related discussion to emphasize the proposed system's value for cybersecurity and explorations of its mean performance. Additionally, the study proposes further research based on the issues identified and enclosed in the chapter, as well as applicable recommendations to improve the system and cybersecurity on the whole. Thus, the conclusion chapter's cybersecurity implication includes recommendations to improve detection tools and defenses in cybersecurity to withstand modern cyber threats, contributing to the development of a safe online environment.

Chapter 2

Literature Review

2.1 Background

Cybersecurity continues to be a pressing issue on the digital frontier, as cyberattacks such as phishing become increasingly sophisticated. Among many attack paths utilized malurls have emerged as one of the most common means to deliver a phish or malware program. The literature indicates that traditional methods of detection are no longer adequate and further, machine learning techniques can disable these threats more effectively. In addition, serverless computing is being seen increasingly for its ability to create more scalable and flexible systems in cybersecurity settings. In this chapter, we trace the history of phishing attacks, introduce both traditional and advanced methods for detecting malware-laden URLs or harmful entities bearing malintentions disguised in net code, look closely at just how machine learning can aid in maintaining safety online, and take a long savory bite into whether—and if so how!—serverless computing promises us at last with all this convenience meant for somewhat slower times (if not sometimes only now in its proper immediacy) an infrastructure that is scalable, cost-effective even for development work by students. We also offer case studies and real examples that show how ML and serverless computing can be integrated to create a secure detection system.

2.2 Phishing and Malicious URLs

2.2.1 Evolution of Phishing Attacks

Originally simply low-level network attack form phishing, now grow up to be exert increasingly cunning attacks mainly by the unnoticed manipulation through social engineering techniques or new technology. In those days, phishing merely entailed scam

mails concentrating on individual users. But today’s phishing is highly individualized, gathering information from social networks as well as other digital platforms which will enable a message to resemble what trick scam victims into clicking on dangerous URLs.

Years later, the attackers learned to bypass conventional detection systems by using makeshift links to disguise their malicious URLs. In other words, these URLs showed up on the outside like normal Internet addresses with abbreviated endings-e.g. dz.ru or co.kr. These are hard for even the old-school detection programs to recognize as threats. As domain generation algorithms began to take hold, the identification issue became more complicated still. As attackers were now able to produce fresh domain names in real-time, blacklists lost their effectiveness [3]. Spear phishing is a new type of phishing, which is particularly targeted and difficult to detect. For example, high value company staff who might be part of this effort are under someone’s careful surveillance before they are made even targets for attack.

2.2.2 Traditional Detection Methods

The traditional techniques for Identifying malicious URLs are blacklisting, heuristic analysis, and signature-based detection. These techniques depend on static details to discover phishing attacks and, as a result, are vulnerable to increasingly dynamic phishing attacks. The disadvantage is that all these methods depend on the static information that exists; hence is limited by its reliability. The three traditional forms are as follows:

1. **Blacklisting:** It works by learning the information of the known malware host URL. The list contains multiple hosts URL. When the URL typed by the user indicates a match, the corresponding search engine console terminates the search for the user. However, it is not applicable to the daily interaction of URL shortening services because short URLs deal with newly generated website content daily. The method is computationally expensive as the time for execution is high [30].
2. **Heuristic Analysis:** The process is to identify unusual behaviors among the websites. Multiple methods for detecting phishing URLs include the likes of using an unusually long URL, the use of unusual top-level domains, the presence of multiple digits of characters, or the use of certain characters such as “@”. There is the possibility of multiple false positives due to the exception that few malicious URLs share the same characteristic as both heuristic analysis frequencies. It is about batch learning on training examples [17].
3. **Heuristic Analysis:** The constant monitoring of the phrases or symbols of malware or malicious URLs which indicate webpages as suspicious is an example of

using predefined rules. The identifiable objects can be webpages as they contain spam comments of some sort. The dictionary of the signs or symbols is responsible for retrieving them. As a result of the system crashes, the signature-based detection technique is required. Finally, the effect of any external factor such as a new kind of spam would thwart the weakness of the above methods [8].

Aspect	Blacklisting	Heuristic Analysis	Signature-Based Detection
Detection Method	Checks against a list of known threats	Analyzes behavior and characteristics	Compares against known malware signatures
Efficiency	Fast for known threats	Slower due to complex pattern matching	Fast for known signatures
Protection Against	Known threats	Zero-day and unknown threats	Known threats
False Positives	Low risk (depends on list quality)	Higher risk due to pattern overlap	Low risk when signature is accurate
Updates Required	Frequently	Not as frequent	Frequently
Strengths	Simple and fast	Can detect unknown variants	Very effective for known threats
Weaknesses	Useless against new threats	Can result in false positives	Cannot detect new or modified threats
Use Cases	Blocking known phishing or malware URLs	Spotting new malware or suspicious behavior	Antivirus and intrusion detection systems

Table 2.1. Comparison of Traditional Detection Techniques

2.2.3 Advanced Detection Techniques

To overcome the shortcomings inherent in traditional approaches, researchers are increasingly using ML and deep learning (DL), characteristics of URLs URL-based indicator of maliciousness. To this day these techniques are still quite mistaken in real time analysis of the URL, so it can detect previously unknown threats.

1. **Machine Learning-Based URL Detection:** By applying ML methods such as decision trees, SVMs and logistic regression to URL detection, has already

done much analysis on not only the lexical structure of the URL but also domain age and WHOIS data. These machine learning based models train themselves with large numbers of both legitimate and malicious URLs, allowing them to spot these suspicious formations according to some rather subtle pattern not visible at all to traditional systems' ways of working [14].

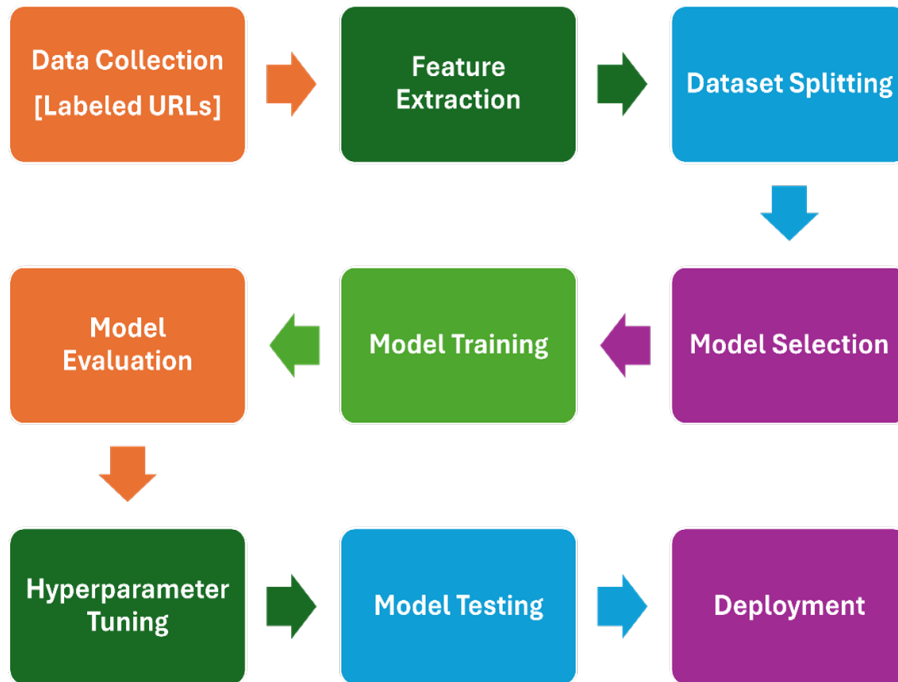


Figure 2.1: Machine Learning-Based URL Detection

2. **Deep Learning Techniques:** As per many of the more recent research work reports, deep learning models have been included in URL detection systems, particularly the CNNs and the RNNs. More specifically, CNNs are better suited to evaluating the lexical characteristics of URLs. On the other hand, RNNs can capture whole sequences of data and McCall, especially where it refers to this, i.e. content available on websites linked to URLs that are suspected in the creation of attacks. These models have shown far better performance levels than the traditional ML technologies in the context of identifying new threats as the existing ones are all known [18].
3. **Natural Language Processing (NLP):** NLP techniques are being integrated with ML models to crack the content of websites connected with URLs. These models, by analyzing text and site structure, can judge whether a URL service constitutes a scam or other type of harmful activity [19].

2.3 Machine Learning and Cybersecurity

2.3.1 Insight into Machine Learning Frameworks

ML has gained widespread adoption among cybersecurity solutions as it can process and analysis high amounts of data. ML-based URL detection models. Several methods of machine learning algorithms are proposed for identifying Malicious URLs.

1. **Random Forests:** Random forests message in when ensemble learning is used to create multiple decision trees during training and output the class that sees (Classification) or mean (regression/mean/mode) prediction of individual tree. This method is used for avoiding overfitting and more performant in cases with a large amount of data or if the dataset is complex.
2. **Gradient Boosting:** The latter class of effective algorithms have become proven the most accurate ones and consequently taking the first place as the most used in standard ML problems competition, such as XGBoost. His iterative feature helps with improving the existing models as saving the failures of the previous version of the development; therefore, making them more proficient in URL finding [3].
3. **Convolutional Neural Networks (CNNs):** There are two major ways of implementing image classification in our URL problem, such as Convolutional Neural Networks and the use of more simple ML algorithms. The models are adjusted to learning more complicated URL-patterns, and as a result, have ultimately shown better performance over standard ML algorithms [14].
4. **Recurrent Neural Networks (RNNs):** In NLP, such models are frequently used, and in this case, they are applied to analyzing URL sequences. RNNs see URLs as sequences of characters and can learn the patterns of “bad” URLs on time, consequently, are the most useful in the fight with unusual attacks, or just as transient attacks [18].

Table 2.2. Comparison and Analysis of ML Algorithms

Aspect	Random Forests	Gradient Boosting	CNNs	RNNs
Data Type	Structured/ tabular data	Structured/ tabular data	Image, spatial data	Sequential data (time- series, text, speech)

Aspect	Random Forests	Gradient Boosting	CNNs	RNNs
Training Time	Fast for small datasets	Slower due to sequential model building	Slow, especially with large datasets	Slow, particularly for long sequences
Handling Overfitting	Better than decision trees (ensemble)	Prone to overfitting if not tuned	Can overfit on small data; regularization required	Prone to vanishing gradients in long sequences
Computational Complexity	Moderate (depends on the number of trees)	High due to sequential nature	High (requires GPUs for large datasets)	High, especially with LSTMs/-GRUs
Performance on Structured Data	High (good baseline model)	High (can outperform Random Forests)	Poor	Poor
Performance on Image Data	Poor	Poor	Excellent	Poor
Performance on Sequential Data	Poor	Poor	Poor	Excellent
Model Interpretability	High (easy to interpret)	Medium (requires more tuning)	Low (deep networks are hard to interpret)	Low (LSTMs/GRUs are black-box models)

2.3.2 Feature Engineering for URL Identification

Feature selection is fundamentally important in a machine learning model's success or failure. Features are often driven by human intuition and hard-won knowledge. Many modern predictive models—in particular nonlinear models—are designed to work even if it is complete or nearly lists all useful features on which they operate. As far as determining the likely danger presented by a given URL is concerned, URL feature engineering means you are extracting relevant characteristics from URLs and associated data. URL detection involves extracting relevant characteristics from URLs and associated data to identify potential malicious activity.. Common features used in URL detection include:

1. **Lexical Features:** These are some of the simplest and most straightforward features to examine. This includes the URL's length, whether it uses an uncommon TLD, and whether it contains special characters. It is particularly effective at identifying phishing URLs, where obfuscation of the domain name is a common occurrence or URL shortening services are used [30].
2. **Host-Based Features:** These are based on data about the domain or server that is hosting the URL. Some examples might include the age of domain registration, available WHOIS data, and the existence of SSL certificates for host-based features. The aim of this type of features is to differentiate between legitimate and outright malicious URLs by examining the behaviour of the website based on the features available about where the site is registered or hosted [17].
3. **Content-Based Features:** These are based on the mined contents of the website that the URL is pointing to, such as the presence of suspicious scripts or iframes. This is extremely effective when trying to discover the work of phishing websites, that frequently look like a regular website, but also host malware which can be identified through the features available on the site [18].

2.3.3 Challenges in Applying Machine Learning to Cybersecurity

Machine learning in cybersecurity applications posed several challenges. Nearly all issues are related to processing and updatability due to the development of attackers' phishing and malware:

- **Data Quality and Availability:** As described, human learning algorithms require large and high-quality datasets. Developing malware and phishing demands an even bigger sequence of data for proper functioning. For example, training human recognition to distinguish potentially malicious URLs from legitimate ones, a labeled dataset of such URLs is required. However, the problem is that this data is not the easiest to obtain, especially in the case of new kinds of phishing schemes [17].
- **Adversarial Attacks:** This issue urges security measures that work properly but in the wrong way. Adversarial attacks are a significant problem with machine learning. If attackers refuse proper human use of tools, at least they should be able to bypass them. An adversarial attack is a situation where the attacker deliberately makes targeted changes to input data, misleading human understanding or specifics of the tool [6].
- **Model Maintenance and Retraining:** Developing cyber threat evolution ensures that attack methods that machine learning knows about today will be

completely different tomorrow. To detect new malware and phishing acts, and to resist old ones by creating new logs, the model must be constantly trained and renewed. Such actions require a lot of computational resources and human resources [18].

2.4 Serverless Computing in Cybersecurity

2.4.1 Overview of Serverless Computing

Serverless computing is a service provided by a cloud computing provider, which permits the deployment of code without the need for managing infrastructure. In this type of architecture, the cloud provider manages and dynamically allocates resources for the client, scaling the application by the minute. This cloud computing model would be particularly useful in the development process of an event-driven application with needs of high fluctuation in use. This includes a significant part of cybersecurity applications, as the workload may vary over intensely during a day cycle or across a week. Serverless computing in general has numerous benefits and might be considered more secure and effective for cybersecurity applications [11].

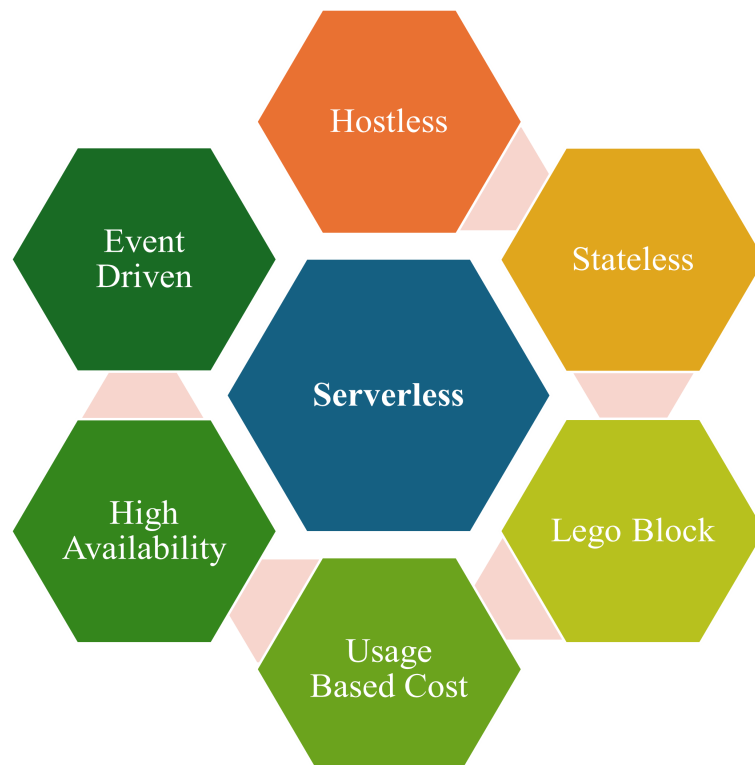


Figure 2.2: Serverless Computing in General

Serverless computing offers several advantages for cybersecurity applications. Among the benefits of serverless computing are scalability, cost efficiency, and lesser operational complexity.

- **Scalability:** Scalability is rather a needed feature of cybersecurity applications, which can expect a high number of attempts and transactions over the use. The applications driven by serverless platforms can handle a great amount of traffic, scaling their resources effectively.
- **Cost Efficiency:** Cost efficiency reduced expenses on infrastructure in the micro-managed programs where users pay only for the resources used and in active use [24].
- **Reduced Operational Complexity:** Less operational complexity for cybersecurity and network security applications, where the need for infrastructure management is eliminated and the developers can focus on the improvement and the development of the security applications [25].

2.4.2 Benefits of Serverless Computing for Cybersecurity

Serverless computing can be advantageous when developing recording integration solutions for utilities as with it, developers could focus on solving business problems using already provided instruments and tools. For instance, instead of concentrating on server scaling and configuring security settings, a security developer could work on detecting malicious URLs and have more time to refine the results. Since the serverless architecture automatically scales with the incoming requests, the high traffic during periods when phishing attacks are the most likely will not overload the system. The serverless platform includes diverse security tools such as encryption and automatic patching and offers integration with other cloud services like databases and machine learning APIs. Thus, solutions for utility customer recording could benefit from such an approach [11].

2.4.3 Challenges and Limitations of Serverless Computing

Below is a list of difficulties connected with serverless computing:

- **Cold Start Latency:** A Serverless computing system requires a cloud provider to allocate the resources and boot the function's environment at the first invocation of a function. It increases the latency of getting the function running. Moreover, if there is much time between the invocation of functions, there is the possibility that the cloud provider may stop the instance that ran the functions. In this situation, there will be a time period when the system needs to start all

over again. This time period is also referred. It is not desirable in time-sensitive applications, especially when it comes to cybersecurity [11].

- **Security Risks:** There are some categories of attacks that target infrastructure of types of the action itself. It concerns the following types of attacks: code injection, privilege escalation, Denial-of- Wallet: the model is based on the system being paid due to the number of invocations of the functions. Attackers send too many requests to the system and in this way, the owner of the system needs to pay for it [6].

2.5 Integrating Machine Learning with Serverless Computing

2.5.1 Architectural Considerations

When blending machine learning with serverless computing, careful planning is essential to ensure performance and cost-effectiveness. On one side, it is well known that serverless platforms are most efficient for hosting very lightweight ML models and perform real-time inference, while model training can be done on a more efficient infrastructure, which usually involves more resources. A paper titled Serving DL models on a serverless platform was highly effective in its implementation of a credit card fraud detection system through using the proposed architecture. It demonstrated a high level of security and efficiency of this system in performing an event-driven cybersecurity task in real time. I have also recently read a similar paper that concluded that the developed ML algorithm was effective in real-time object recognition and tracking where the platform was the star of the show [5].

2.5.2 Case Studies and Applications

In cyber security, case studies show how machine learning's successful integration with serverless computing. One example is the HealthFaaS system, a service for real time health monitoring. It's designed with a serverless architecture to let scales up or down dynamically based on what kind of demand there is from its users [11]. In another example, systems for credit card fraud detection have been moved onto serverless platforms. Working like this exposes information that can be analyzed in real time and can cut down on fraud [5].

Table 2.3. Literature Review Summary

Paper Title	Methods	Algorithms	Results	Future Work	Limitations
Survey on Malicious URL Detection Techniques [3]	Survey of blacklist-ing, rules-based, ML, and DL methods	SVM, Random Forest, CNN	Accuracy: Up to 99% for some ML models	Integrating new URL features, enhance real-time performance	Content-based features not utilized
Performance Evaluation of Serverless Edge Computing for ML [16]	Serverless frameworks (Kubeless, OpenFaaS)	CNN, LSTM	Kubeless: Lowest response time, Fission struggled	Optimizing autoscaling and reducing latency	Auto scaling of modules not tested
URL Based Malicious Activity Detection Using ML [8]	Comparative analysis of ML algorithms on URL data	Random Forest, LightGBM, XGBoost	Random Forest accuracy: 94.5%, XGBoost: 93.2%	Incorporating dynamic features for improved detection	High processing time; no use of serverless computing
ML Supported Malicious URL Detection [19]	Feature engineering and ML classification	SVM, Random Forest	Random Forest accuracy: 96%, SVM: 93%	Including dynamic analysis of URL content/behavior	Only lexical features used; content features missing
Hybrid Approach for Malicious URL Detection [10]	Hybrid model combining blocklists, ML, and DL	SVM, CNN	Accuracy: 98%, outperformed blacklist-only methods	Reducing false positives, improving real-time capabilities	High processing time; lacks serverless computing for scalability

Malicious Short URLs Detection Technique [30]	Rule-based feature extraction and ML classification	Random Forest, SVM	Random Forest accuracy: 92.3%, SVM: 89.8%	Focus on URL shortening services and evasion techniques	No content-based features; lacks scalability analysis
New Heuristics Method for Malicious URL Detection [17]	Heuristics-based feature extraction combined with ML	Decision Tree, Random Forest	Heuristics + Random Forest accuracy: 93%	Optimization for scalability and complex threats	High time of processing; lacks serverless computing
Malicious URL Linkage Analysis and Common Pattern Discovery [15]	Linkage and pattern analysis using statistical methods	Statistical analysis	No specific accuracy provided	Integrating patterns with ML for predictive models	Does not focus on ML techniques; lacks real-time applicability
CNN-Based Model for Detecting Malicious URLs [14]	CNN-based classification	CNN	CNN accuracy: 99.2%	Focus on handling complex URL structures	High processing time; no content-based or serverless approach

2.6 Summary

The chapter supplies us with a wide literature review on the evolution of phishing attacks and the increasing complexity of malicious URLs. It also has addressed the applicable limitations of older detection methods, such as blacklisting & signature-based detection, which are incapable of keeping pace with the growing complexity of cyber threats. For these reasons, new detection methods had to be developed, such

as ML and DL, able to provide real-time analysis and identify previously unknown sources of danger. It also noted that serverless computing can be quite beneficial as far as this type of activity concerned because it was deemed more scalable, cost-effective, and capable of handling real-time data processing. By combining serverless and ML systems, it is possible to develop more adaptive and thus safer systems, which would be followed by a number of appropriate case studies.

Chapter 3

Methodology

3.1 Introduction

This chapter will give a comprehensive, methodological analysis of how to design and implement a ML system for identifying malicious URLs running on serverless computing platforms. This system not only draws on lexical features such as URL structure or physical properties of web hosts is traversable by our system, but also uses the content-based scores that come from mining all corresponding web page segments, to track and catch unsafe URLs leading off towards fake web shops downloading malware and other potential threats. The methodology involves a number of stages. One of the most important steps is the collection of the dataset and in-depth feature engineering work which follows. With just over 5 dozen or so extracted features of this kind, the process involves looking for anything that could be any kind hint anything like an fingerprint perhaps. Some ranges of these characteristics and fingerprinting elements range from obvious URL properties to the more advanced content analysis based on structure and web page components.

System was install in traditional local mode, entirely on a serverless architecture and by combining the above two methods, achieve still higher cost of availability. By deploying the solution on scalable, real-time processing-capable cloud-based serverless architecture (Google Cloud Functions etc) this twitch is achieved. Yet this serverless framework lets you scale upwards and downwards dynamically according to workload and needs no manual resource management for high volumes.

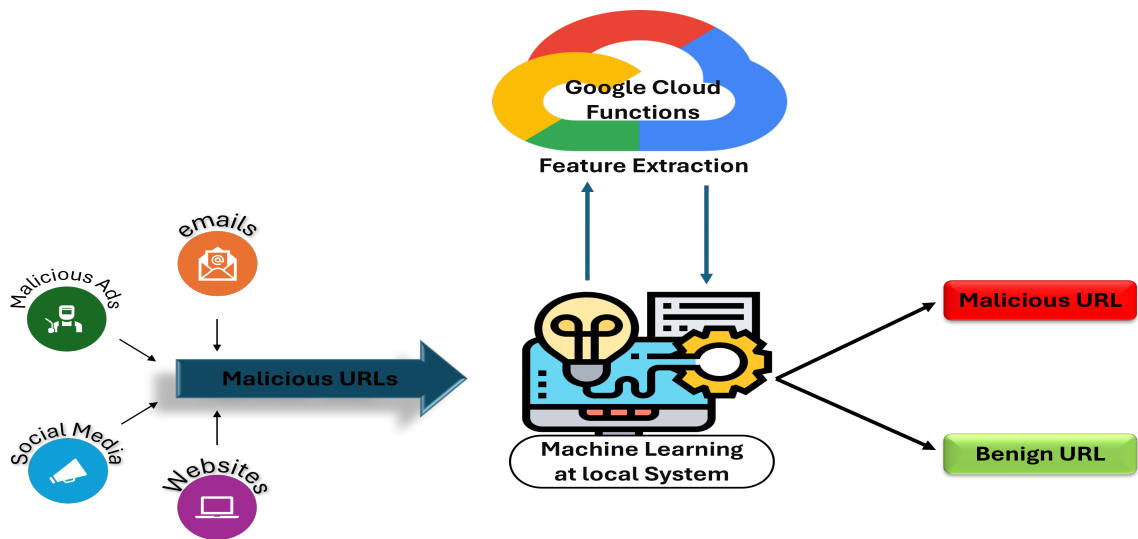


Figure 3.1: General Architecture

Multiple machine learning models were also trained and tested to identify malicious URLs. Methods such as boosting and ensemble models were among those implemented and then subjected to strict, cross-validated evaluations for accuracy and generalizability in a variety of situations. Cross-validation was important for preventing overfitting and ensuring that the models performed well across a variety of datasets, including new real-world data sets that had never been seen before.

3.2 Research Design

3.2.1 Overview of the Research Process

The research route makes use of these three key stages to approach: building Methodology. Deploying Methodology. Evaluation machine learning-based malicious URL detection system:

1. **Data Collection:** Vast amount of data were gathered from many sources including the live base metal repositories PhishTank [21] and VirusTotal [31], as well as various other live bases like OpenPhish [20]. Benign URLs came from established repositories to ensure balance in the dataset.
2. **Data Cleaning:** Only the live URL from hundreds of thousands to millions. In this way one may gradually weed out some bad choices; moreover it is also certain that one can ensure correct analysis results through model.

3. **Feature Engineering:** With 54 features in all, both lexical and content- based ones were extracted from the URLs and their hosting web pages. Some features were generated from URL structures, whereas others originated in the text of such pages.
4. **Model Development:** Then I tried several machine learning methods, including boosting and ensemble modelling, with this feature-engineered dataset. Although I aimed at increasing correct detection rates still further while reducing false alarms down to some acceptable minimum level, Multiple models were added to maximize chances that the system could also achieve a high level of precision in identifying what is good or bad in an unknown web page-even if at first glance it looks totally innocent.
5. **System Deployment:** The core malicious URL detection system is implemented and run on a local system, which gave full control over model training, classification and performance evaluation. To enable the extensive demands of feature extraction, Google Cloud Functions [7] were used. By allocating only feature extraction to cloud-based functions, the system realized an almost perfect balance of scalability in preprocessing-while still giving good results locally. This hybrid approach saved serverless computing for power demanding chores, while continuing with using the local environment for better control of system processes and resources.
6. **Performance Evaluation:** The performance of the machine learning models was evaluated with various key metrics, such as accuracy rate vs. recall rate. K-fold cross-validation was applied to test robustness and generalizability, which meant that results would probably continue to hold true across different datasets and not just for whatever set you happened get first (or last!).

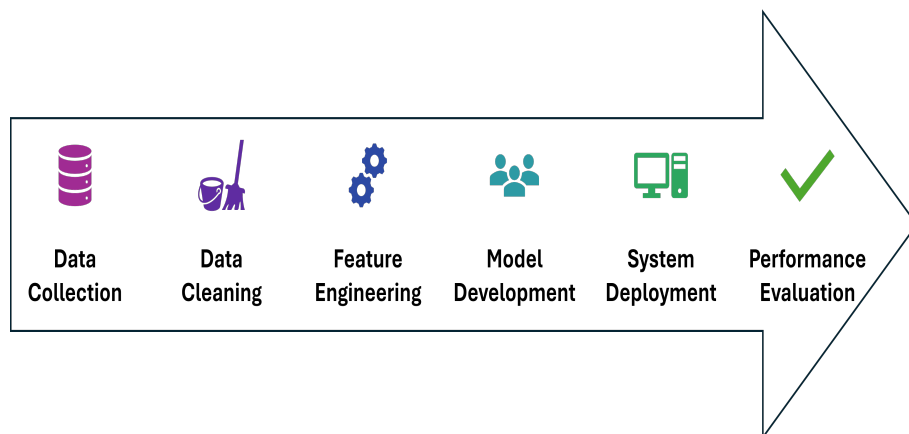


Figure 3.2: Google Cloud Function and Storage

3.2.2 Data Collection

The dataset used in this study has been carefully prepared from multiple sources to ensure diversity, reliability and comprehensive coverage of both malicious and benign URLs [13]. An objective of this research was to build a dataset large enough to be representative and balanced to train a machine learning model detecting malicious URLs with sufficient accuracy. The sources of the malicious URLs were trusted and reputable platforms specializing in cyber security and phishing discovery. The sources of the confirmed and far from being malicious URLs were various, and it is appropriate to list only several of the most representative ones.

3.2.2.1 Malicious URLs

Malicious URLs were obtained from trusted, reputable platforms that specialize in cybersecurity and phishing detection. The following sources were utilized to compile a rich collection of confirmed malicious URLs:

- **PhishTank:** An open-source community-driven platform that collects the reports of phishing activities and verifies phish URLs. Due to the open nature of this project, the number of users submitting the reports, and the methodology of verification, this resource is one of the most reliable real-time sources of confirmed phishing URLs [21].
- **VirusTotal:** A web-based service that allows checks for malicious URLs using multiple antivirus engines, and other tools for scanning websites. Each URL check returns the type of threat found – malware, phishing, or any other suspicious activities, – and provides a detailed report: “VirusTotal is a free service, and can handle URLs, files up to 128MB in size, and even shortened URLs [31].
- **OpenPhish:** A real-time service that detects phishing URLs and constantly monitors them in order to keep a current list of the malicious URLs [20].

3.2.2.2 Benign URLs

In order to help guarantee that the data set would be balanced and complete, benign URLs were taken from very recognized web repositories which have long held trust. Selecting these URLs was based on their credibility and verified legitimacy. Inclusion of clean URLs is essential if you expect your model to accurately distinguish between safe URLs and dangerous ones. Examples of sources for clean URLs include:

- **Safe Browsing Lists:** Trusted sites that maintain clean and safe websites.
- **Whitelisted Domains:** Domains known to host legitimate content like government web sites, well established enterprises, verified online service providers.

3.2.3 Data Cleaning and Preprocessing

A data cleaning and preprocessing phase was carried out to ensure that the dataset was of high quality and could be used to train and test a machine learning model. This step was pivotal because it minimized the quantity of noise in the data, reduced detrimental sources of bias, and enhanced the model's performance and generalizability. Data cleaning was performed as follows:

3.2.3.1 Live URL Validation

The first step in data cleaning was to ensure that the URLs in the raw dataset were functional. This is an integral step because the presence of broken or inoperative URLs in the dataset will evidence being used to train the model, and introduces large amounts of noise into the data, thus, severely weakening the functionality and the robustness of the model. Thus, the following steps were taken to validate the functionality of the URLs:

- **HTTP Status Code Check:** Each URL was checked individually to ascertain its functional status. To facilitate this, the requests library in Python was used to access each URL via the HTTP protocol. URLs that returned an HTTP status code of 200 were considered to be functional and valid, and were retained for the next step of the analysis [26].
- **Filtering Out Inactive URLs:** URLs returning status codes of 404 or 403, or a status code of 500 and other error codes, were discarded. Thus, the final dataset or module was used to train machine learning models that were free of broken and unusable URLs.

This process ensured that the machine learning models were trained only on active URLs, improving both the quality and reliability of the dataset.

3.2.3.2 Duplicate Removal

As repeated instances of the same URLs are present into the dataset hence it could also generate bias into the dataset because a single URL could appear multiple time and it could highly influence model to learn from those ones. It was rectified as:

- **Duplicate Detection:** I checked the URL file which contains all the links for the webpages against duplicates. As it was identified all URLs which appear more than once and hence removed from the dataset.

- **Impact of Duplicate Removal:** Due to its removal each the URL whereby the index which holds the URL has duplicates, eliminated. I took this step to make sure my dataset is not skewed towards a single or few URLs because if a URL is repeated greatly so it will be learned greatly by the model and this will be a bias for the model. Thus the model will learn from a wider range of URL.

By eliminating duplicates, the model could focus on learning unique patterns and features in the URLs rather than being skewed by repeated examples.

3.2.3.3 Balancing the Dataset

In machine learning systems, imbalanced datasets, or datasets with one class being overtaking another in terms of their number representation in the dataset, can become the reason for the poor performance of the model on the completion of the task on the under-represented class. In our case, if an imbalance is found between the benign and malicious URLs numbers, the system might develop the readiness to label any site as benign, thus missing the majority of malicious URLs in the testing sets. In order to prevent this, the following step was included:

- **Class Distribution Check:** At this step, the number of examples for both benign and malicious URL classes is checked in the dataset; if there was an imbalance, the result of this step would be as follows.
- **Equalization of Class Distribution:** In case of an imbalance, different methods of balancing the class representations across the dataset, e.g. under sampling of benign URL class, or oversampling of malicious URL class, are utilized to remove such imbalance and ensure unbiased representations of both classes. This step will lead to the restriction of the imbalance effect, enabling the resulting model to process malicious URLs without the predisposition to accept all URLs as benign.

Balancing the dataset improved the model's ability to accurately detect malicious URLs without being biased toward benign ones, resulting in a more robust system.

3.3 Feature Engineering

Feature engineering was vital to achieving the development of a high performing detection system capable of accurately identifying malicious URLs. This included extracting meaningful factors from the URL and the content of the web page that serve as signals to distinguish between benign and malicious behavior. This part threatens the processing system because malicious content of webpages was need to be analyzed. To

address this issue, google cloud function (serverless computing) was used. It provided sandbox like environment and extracted features securely. It also expedited the process by extracting features rapidly by using large resources of serverless computing. Features were extracted into two main categories, lexical features, and content-based features. Total 54 features were extracted, detail is as under:-

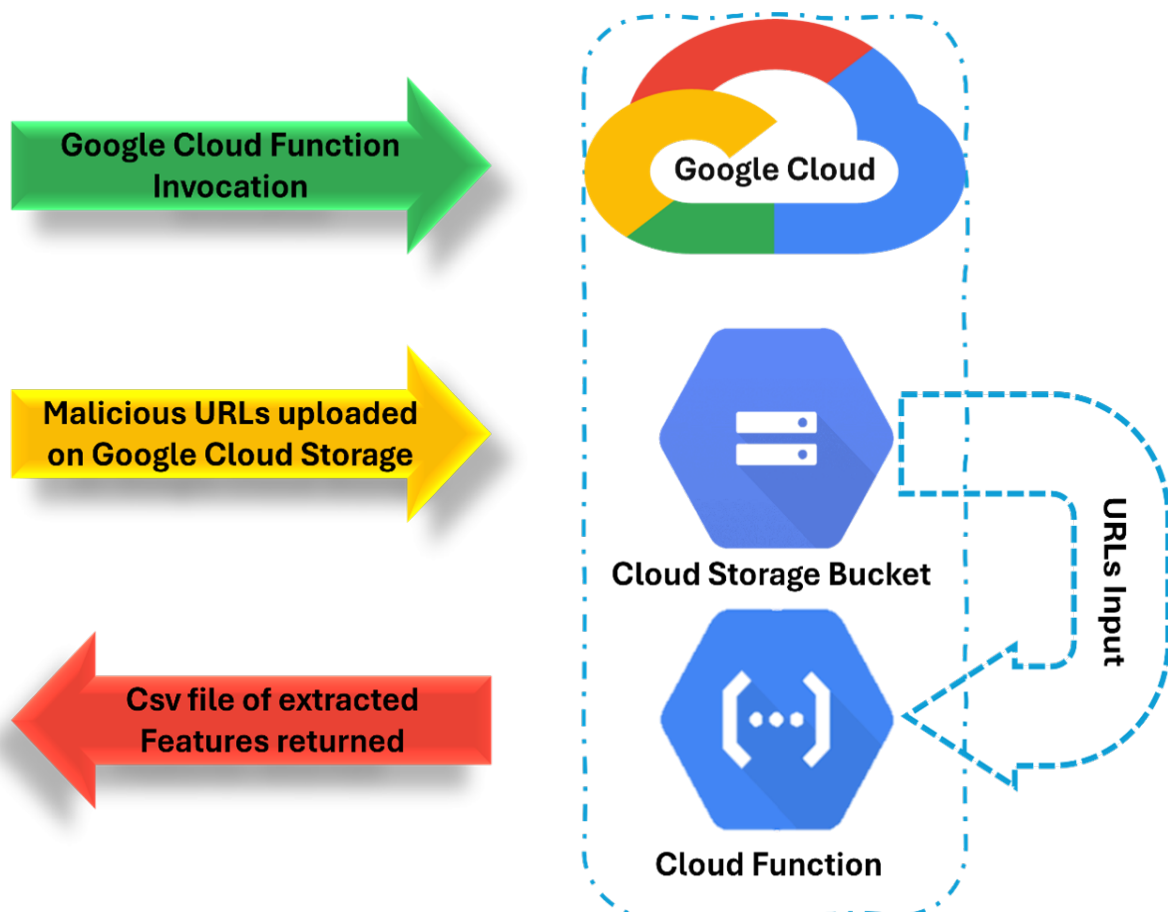


Figure 3.3: Google Cloud Function and Storage

3.3.1 Lexical Features

Lexical features were extracted solely from the URL structure without needing to access the webpage's content [29]. These features provided valuable information regarding the URL's construction and potential indicators of malicious intent. Below are the key lexical features and their explanations.

1. **Length of URL:** Total number of characters in the URL.

2. **Number of Dots:** Number of periods (.) in the URL, indicating subdomains or complex structures.
3. **Number of Slashes:** The count of slashes (/) in the URL, often indicating deeper paths.
4. **Number of Params:** Number of query parameters in the URL (after "?").
5. **Has https:** Whether the URL uses HTTPS for secure communication (True/False).
6. **Has www:** Whether the URL contains "www" (True/False).
7. **Domain Length:** The total number of characters in the domain name.
8. **Path Length:** The number of characters in the path section of the URL.
9. **Subdomain Length:** The length of the subdomain part of the URL.
10. **Is IP:** Whether the URL contains an IP address instead of a domain name (True/False).
11. **Number of Digits:** Number of numeric characters in the URL.
12. **Has Query:** Whether the URL contains a query string (True/False).
13. **Number of Special Chars in URL:** Number of special characters (e.g., @, %, &) in the URL.
14. **Number of Uppercase in URL:** Number of uppercase letters in the URL.
15. **Shortened URL:** Whether the URL is shortened (True/False).
16. **URL Entropy:** Measure of randomness in the URL string (higher values indicate suspiciousness).
17. **Top Level Domain:** The top-level domain (e.g., .com, .org).
18. **URL Depth:** Number of directories or slashes in the URL path.
19. **Has @ Symbol:** Whether the URL contains an "@" symbol, often used in phishing URLs (True/False).
20. **Has Hash Symbol:** Whether the URL contains a "#" symbol (True/False).
21. **Has Subdomain:** Whether the URL contains a subdomain (True/False).
22. **Has Port:** Whether the URL specifies a port number (True/False).

23. **Number of Alphabets in URL:** Number of alphabetic characters in the URL.
24. **URL Encoded:** Whether the URL is encoded (True/False).
25. **Is Long URL:** Whether the URL length exceeds a certain threshold (True/False). (e.g., 75 characters).

3.3.2 Content-Based Features

Content-based features were derived by analyzing the actual content of the webpage to which the URL points [27]. These features provided insights into the nature of the webpage and its potential to be malicious, helping to identify phishing or harmful websites by examining the elements present on the page. Below are the key content-based features:

1. **Number of Images:** Number of images present on the webpage associated with the URL.
2. **Number of Links:** Number of hyperlinks on the webpage.
3. **Number of Words:** Total number of words on the webpage.
4. **Number of Headings:** Number of heading tags (e.g., <h1>, <h2>) on the webpage.
5. **Number of Meta:** Number of meta tags (e.g., <meta>) on the webpage.
6. **Has Meta Description:** Whether the webpage includes a meta description tag (True/False).
7. **Meta Description Length:** Length of the content in the meta description.
8. **Has Meta Keywords:** Whether the webpage contains meta keywords (True/False).
9. **Meta Keywords Length:** Length of the content in the meta keywords tag.
10. **Title Length:** Number of characters in the webpage's title.
11. **Number of Scripts:** Number of script tags (e.g., <script>) on the webpage.
12. **Number of Styles:** Number of style tags (e.g., <style>) or inline styles.
13. **Number of iframes:** Number of iframes (e.g., <iframe>) on the webpage.
14. **Number of Buttons:** Number of buttons on the webpage.

15. **Number of Forms:** Number of form elements (e.g., `<form>`) on the webpage.
16. **Number of Inputs:** Number of input fields on the webpage.
17. **Average Word Length:** Average length of words on the webpage.
18. **Number of Uppercase Words:** Number of uppercase words on the webpage.
19. **Number of Digits in Text:** Number of numeric characters within the webpage's content.
20. **Number of Special Chars:** Number of special characters in the webpage content.
21. **Has Video:** Whether the webpage contains video elements (True/False).
22. **Number of Bold:** Number of bold tags (e.g., ``, ``) on the webpage.
23. **Number of Italic:** Number of italic tags (e.g., `<i>`, ``) on the webpage.
24. **Number of Tables:** Number of table elements (e.g., `<table>`) on the webpage.
25. **Number of Lists:** Number of list elements (e.g., ``, ``) on the webpage.
26. **Has Canonical Tag:** Whether the webpage includes a canonical tag (True/False).
27. **Has Favicon:** Whether the webpage includes a favicon (True/False).
28. **Number of Embeds:** Number of embedded elements (e.g., `<embed>`) on the webpage.
29. **Has Open Graph:** Whether the webpage includes Open Graph meta tags for social media sharing (True/False).

3.4 Machine Learning Model Development

The newly prepared dataset was used to train different machine learning models. The conclusions were drawn based on the models' corresponding points of minimums. For each algorithm, there is a certain aspect of its classification performance on the basis of which it can be both an asset and a potential drawback. Below is a detailed analysis of the algorithms, considering the results presented above.

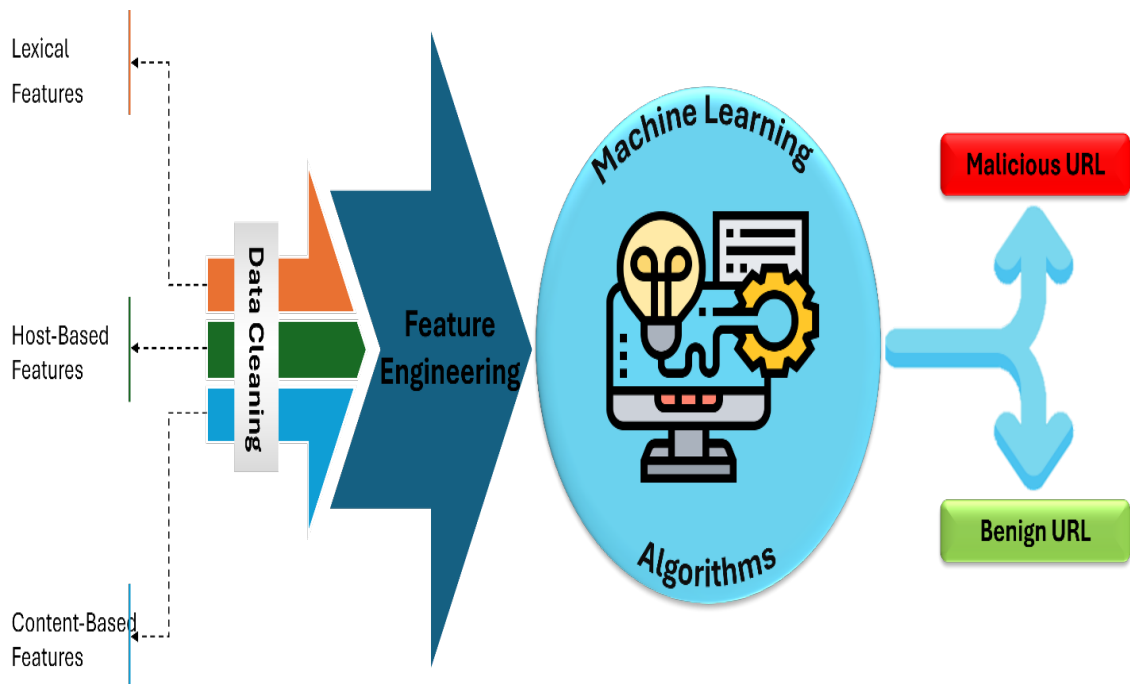


Figure 3.4: Machine Learning Model

3.4.1 AdaBoost

AdaBoost is short for adaptive boosting. It is a type of ensemble learning method that focuses on building a strong classifier by aggregating multiple weak classifiers [4]. AdaBoost is conducted in an iterative way during which the training model of every round of iteration pays more attention to those training samples that were misclassified by the previous round's weak classifiers.

Specifically, it shifts the focus onto the samples that are hard to classify by increasing the weights of the misclassified samples, pushing the next weak classifier to pay more attention to it. The final prediction is made depending on the aggregate of the weighted prediction of all weak classifiers who are the learners of weak classifiers.

AdaBoost is particularly suitable to handle the binary classification and the boosting mechanism of AdaBoost benefits accuracy by focusing on hard instances in every round. In addition, it has an adaptive feature which indicates that the training of each weak classifier is based on the learning experience from the previous classifiers.

- **Weak Classifiers:** The weak learners used in AdaBoost normally are simple decision stumps which are single-level decision trees.

- **Boosting Mechanism:** AdaBoost is a type of ensemble learning algorithm that trains the weak learners sequentially with the sample weights tune up in every round.
- **Final Prediction:** Every weak classifier gives a prediction for which the results are then combined into a weighted sum with different capability classifiers weigh differently.

3.4.2 XGBoost

XGBoost, or Extreme Gradient Boosting, is an advanced implementation of the gradient boosting method. It is considered one of the best algorithms due to its efficiency, scalability, and high performance in many ML problems [32]. Like AdaBoost, XGBoost utilizes an ensemble of sequential decision trees. However, it applies the gradient boosting approach, as every new tree focuses on reducing the errors of the previous trees, based on the greedy algorithm that minimizes a differentiable loss function. On the other hand, the advantages of XGBoost include more features and the capacity to handle data imbalance.

The other main strengths of XGBoost are: missing data handling, regularization support, parallel computation use, and high performance that makes it one of the widest-used algorithms for structured data and classification tasks.

- **Gradient Boosting:** every new decision tree seeks to predict the mistakes of the previous trees, known as residual errors.
- **Regularization:** XGBoost applies both L1 and L2 regularization methods, or lasso and ridge penalties, to penalize large weights and, therefore, control the occurrence of overfitting.
- **Handling Missing Values:** the algorithm uses the differentiation method of learning which direction to go if the data is missed, or the splits should be done, taking into account the missing values.

In addition, XGBoost applies other techniques, such as shrinkage or the way of reducing the importance of every tree by a scaling factor, column subsampling, where the algorithm uses not all but a random subset of the available features to build each new tree, and heatrows or the application of filters to focus only on the most growing leaves.

3.4.3 Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training. Each tree is built from a random subset of the training data and a

random subset of the features. Finally, the predictions from all the trees are aggregated to make the final prediction, usually through voting for classification. Random Forest is designed to reduce overfitting by computing the means of many decision trees estimators, where each tree is slightly different due to the application of random sample on both samples of data and features to train the trees [22]. Consequently, Random Forest is not specifically tuned to real datasets and is very robust against noise. The two key mechanics of Random Forest are.

- Bagging: Random Forest uses bagging; this means creating multiple datasets through re-sampling. The re-sampling is done by randomly choosing training data with replacement from the original set and generating a dataset. A decision tree will be trained on every dataset.
- Random Feature Selection: At each node, the algorithm selects a random set of features to make a decision, leading to development of decision trees with different features. This will reduce the correlation between the trees. By averaging the prediction and using Majority for classification the final Random Forest prediction is made.

3.4.4 Extra Trees

Another Random Forest classifier is the Extra Trees, which is more random. It applies extra randomization to the tree-building process. The Extra Trees method selects the random threshold on which to split its nodes instead of selecting the optimal and best threshold [9]. Therefore, the primary sources of randomness in Extra Trees are the following:

- Randomized Node Splitting: It does not look for the best-value split along each feature but selects a random one.
- Feature Randomization: It selects random features in each node.
- Extra Trees has a slightly better generalization performance on specific data sets. In practice, it reduces the high variance of Random Forest. The Extra Tree method is widely used with the help of `sklearn.ensemble` library.

3.4.5 K-Nearest Neighbors (KNN)

K-Nearest Neighbors KNN is a non-parametric, simple classification algorithm. The classification is done such that the classes are assigned to the samples that are in close proximity to other points in the feature space. The sample's class is determined by a majority vote of the samples that are closest to it. Here, k is the user-defined

parameter which determines the number of nearest neighbors to consider. The user-defined parameter is k which determines the number of samples to be considered to classify a given instance [1]. KNN is an intuitive classifier but is computationally expensive as it needs to compute the distance between every test sample to every sample in the training dataset. The performance of KNN is sensitive to the choice of k and the distance metric, for example, Euclidean distance is one of the common distance metrics. KNN is exclusively used for numerical prediction through regression.

- Distance-Based Voting: KNN assigns the class label based on the majority class for the nearest neighbors in the feature space.
- Non-Parametric: KNN has the advantage that it needs to assume no underlying distribution for the data, thus, it can be used in a variety of classification tasks.

One of the challenges is that the KNN is a slow algorithm especially for the large dataset because it needs to compute the distance of every training instance with the test instance.

3.4.6 Stochastic Gradient Descent (SGD) Classifier

Stochastic Gradient Descent is an optimization technique used to train linear classifiers, such as Support Vector Machines and logistic regression. The method consists of computing the gradient of the loss function for a random mini-batch of the data instead of computing the gradient for the entire dataset. As a result, SGD is faster and more scalable for large datasets. SGD is highly computationally efficient, but due to its speed, it is often sensitive to hyperparameters, especially the learning rate and the regularization terms [23]. This method is only suitable for datasets that are linearly separable and usually requires performing feature scaling to ensure its good performance.

- Gradient Descent: The Gradient Descent method iteratively updates the model parameters by moving them in the direction which minimizes the loss function and does so using only a subset of the data, which is characteristic of many types of gradient descent. One of the few exceptions is so-called “full-batch” gradient descent, where the method uses the entire dataset to estimate the gradient.
- Regularization: The SGD can also support a number of regularization techniques to prevent overfitting. It is often used with L2 or L1 regularization, which are also known as ridge and lasso respectively. Normally, it is a very commonly used technique for very large-scale machine learning tasks, because it is very fast and computationally efficient, although due to the number of hyperparameters it may need some amount of time to adjust.

SGD is widely used for large-scale machine learning tasks due to its speed and efficiency, though it requires careful tuning of hyperparameters to achieve optimal performance.

3.4.7 Naive Bayes

Naive Bayes classifier is a probabilistic model adopted in machine learning founded on the basis of Bayes' theorem. Though it assumes that all the features are independent, which might not be true, hence making it naive, the model performs quite well in a good number of classification instances, more so in text classification or categorical data. It performs this task by evaluating all the posterior probabilities of each class assigned, given the features [2]. The priors of the class and the likelihood of these features are then taken into consideration, while operating under the knowledge of the conditional class independence of the features.

Naive Bayes works by calculating the posterior probability of a class given the features, using the prior probabilities of the class and the likelihood of the features, under the assumption of feature independence.

- **Bayesian Theorem:** It relies on Bayes' theorem to evaluate the probability of each class, and the class that comprises the highest posterior probability is then returned.
- **Independence Assumption:** Naive Bayes classifier assumes that the data features are all independent when restricted with the class label. Therefore, this will make it easy to calculate the probabilities, while making a rather unrealistic prediction regarding the features.

Naive Bayes performs well in data with a high dimension such as text classification, and it is also computationally efficient; however, when these features are correlated, the model might not be very useful in data.

Table 3.1. Comparison of Machine Learning Algorithms

Algorithm	Strengths	Weaknesses	Performance on Large Data	Performance on Small Data	Overfitting
XGBoost	Highly accurate, handles large data	Requires careful tuning	Excellent for large, complex datasets	Works fine but better for large data	Can overfit, but manageable
Naive Bayes	Simple, fast, works well on small data	Makes naive independence assumptions	Struggles with very large datasets	Performs well on small datasets	Prone to underfitting

Algorithm	Strengths	Weaknesses	Performance on Large Data	Performance on Small Data	Overfitting
SGD Classifier	Efficient for large datasets	Sensitive to feature scaling	Performs well for large, sparse data	Performs well on large datasets	Low risk if tuned properly
K-Nearest Neighbors (KNN)	Simple and intuitive for small data	Slow for large datasets, sensitive to outliers	Not efficient, scales poorly with large datasets	Good for small datasets	Low risk with proper tuning
Extra Trees	Less prone to overfitting, fast predictions	Computationally expensive	Performs well but requires more resources	Performs well on small datasets	Less prone to overfitting
AdaBoost	Boosts weak learners' performance	Prone to overfitting with noisy data	Performs well but slower to train	Works well on small data but can overfit	Can overfit with noisy data
Random Forest	Reduces overfitting, good with unbalanced data	Requires more memory and computational resources	Good for large datasets	Performs well on small datasets	Reduces overfitting
Decision Tree	Easy to interpret, handles non-linear data	Prone to overfitting	Tends to overfit on large datasets	Good for small datasets	Highly prone to overfitting

3.5 System Deployment

To deploy the system for detection of the malicious URL, I have chosen to have the model implementation in both local and serverless environment to optimize the ability to scale, performance, and resource utilization. At the same time, the feature extraction

part was deployed on Google Cloud Functions based on serverless architecture to ensure better flexibility, scalability, and efficiency in resource utilization without taking care of the underlying infrastructure. Hence, the system uses a combination of local and serverless environments to ensure a balance between flexibility and better scalability.

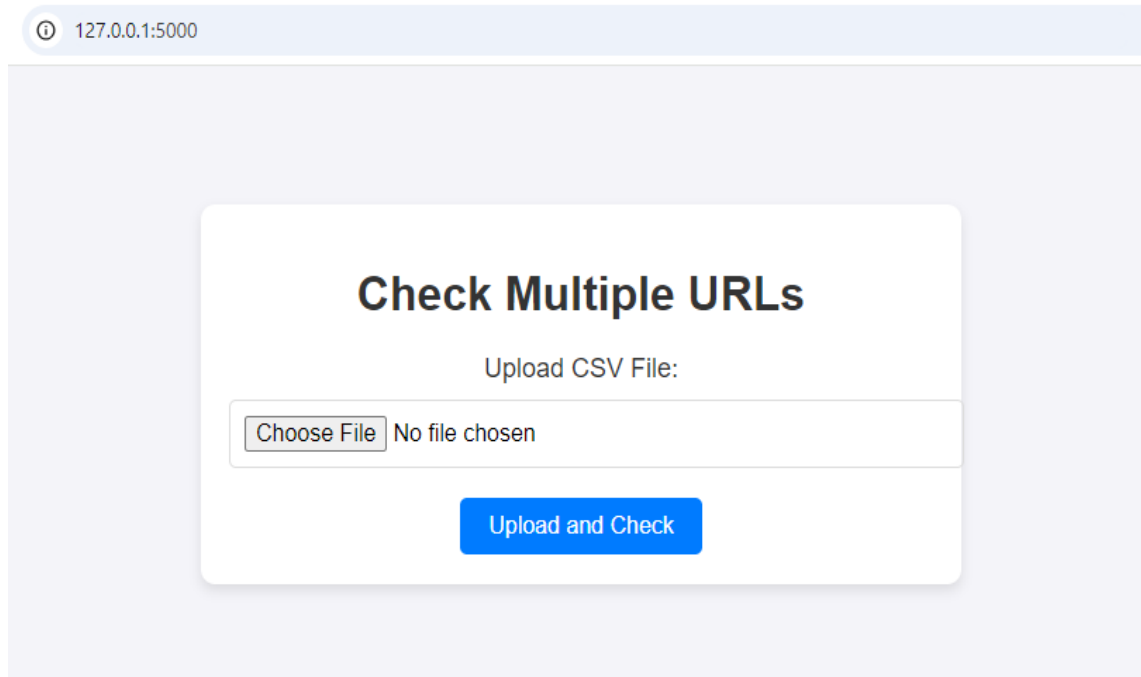


Figure 3.5: GUI Interface of Web based Flask Applications

3.5.1 Local System Deployment

The deployment of the localization of the majority of the detection system was made to ensure complete control over model training, classification, and performance evaluation. Some benefits were accomplished from the deployment of the core of the current detection system as follows:

- **Full Computational Control:** The complete hardware and software control enabled us to carry out more computationally demanding machine learning models, ensure real-time analysis and control the flow of the process without any other system to rely on.
- **Efficient Resource Management:** Model training and evaluation regarding adequate computational resources were executed in the local environment that was especially helpful when having high-level computational processes being executed. At the same time, the latency is lower compared to the more cloud-based approach.

- **Security:** The components related to model training and evaluation were kept more secure in the local system and the risk of having a potential threat was minimized.

3.5.2 Google Cloud Functions for Feature Extraction

Google Cloud Functions for Feature Extraction That stage of the system was realized by using Google Cloud Functions to perform feature extraction, which allowed deployment of lexical and content-based features in real-time. Google Cloud Functions was chosen as a serverless platform because it had several advantages:

- **Automatic Scalability:** Google Cloud Functions automatically expanded according to the number of requests and could cope with large volumes of URLs in real-time without manual intervention.
- **Cost-efficiency:** As a “pay for what you use” service, it only incurred charges when one of the features that it provided had been executed. This dramatically reduced the costs of over-provisioning.
- **No Infrastructure Management:** This approach meant that developers were not required to manage servers or infrastructure, leaving the focus on feature extraction logic itself. Reducing operational complexity while processing efficiently at scale.

3.5.3 System Components in Google Cloud Functions:

- **Google Cloud Functions:** Managed the scalable extraction of both lexical and content-based features from live URLs in real time. This ensured that feature extraction was adaptable to varying traffic conditions without sacrificing performance [28].
- **Cloud Storage (GCS):** Used to store features extracted and intermediate results during the feature extraction process. This allowed seamless integration between functions that extract features and locally deployed components responsible for model training and classification [12].

3.5.4 API Gateway and Routing

- **Google Cloud API Gateway:** Managed incoming HTTP requests and routed them to the appropriate Google Cloud Functions for feature extraction. The API Gateway provided secure, efficient handling of requests from the web interface or external systems, enabling reliable routing to cloud functions.

- **Load Balancing:** Traffic in Cloud Functions was automatically distributed, which meant that all the incoming URLs were processed in real-time without any bottlenecks. This improved system performance overall.

3.5.5 Data Storage and Integration

The results of feature extraction were stored in Google Cloud Storage after the process for subsequent retrieval by locally deployed systems. GCS provided a reliable and scalable storage solution. Organizing the huge volumes of features extracted together with local parts for analytics and classification also was simple.

3.5.6 Integration with Local System

After extracting the features on Google Cloud Functions, we now have the data in the local environment where our machine learning models were deployed. Such a configuration or deployment is based in a hybrid or mix and match model that has both cloud and local implementations. Thus, feature extraction that is more resource-intensive operation including natural language processing occurred in the Cloud platform. On the other hand, the model training and classification took place in the local environment. Not only is the local environment much more flexible, but it also provides the compute power necessary for this strategy.

3.5.7 Hardware Specifications

Following hardware was used for the implementation of the hybrid system:-

- **Local System:** Core i5 1.8 GHz CPU with 8 GB RAM
- **Serverless (Google Cloud Function):** 1 GHz CPU and 256 MB RAM

3.5.8 Key Benefits of Hybrid Deployment

- **Flexibility and Control:** The model training and evaluation were maintained locally to ensure a high degree of flexibility and control over the most important components in the system. The deployment model also provided flexibility to move such key functions to the Cloud at a later stage.
- **Improved Scalability:** Caters to the number of incoming URLs, the feature extraction was offloaded to Google Cloud Functions. As this process was run on a serverless architecture, the feature extraction process automatically scaled up the capacity when there was a need for more resources to handle increasing load.

of incoming URLs. Furthermore, serverless offered the flexibility to allow the deployed system to remain offline as tens or hundreds of thousands of requests had been processed. Such scalability was not possible because of compute costs associated with the servers and manual intervention required to monitor the current traffic and add resources when the current configuration could not handle the traffic.

- **Cost-Effectiveness:** Lastly, cost advantage was realized through the use of the serverless model on the Google Cloud Functions. Since the deployed Google Function only ran when there was a URL to process, the resource consumption was optimized to avoid additional overheads that are attributed by the always-on costs with most server implementations.

3.6 Cross-Validation

Cross-validation is a widely adopted technique used to assess the robustness of machine learning models. It is especially helpful if overfitting may have a detrimental impact on the capability of the model to capture the unseen data. This study utilized the k-fold cross-validation where $k = 5$, meaning that the data was split into five equal subsets. However, it could be done with different numbers of subsets. This method was one of the most accurate solutions to evaluate the performance of the model because, during the process, every observation in this data set is used to train a model and will have an opportunity to be used to test a model.

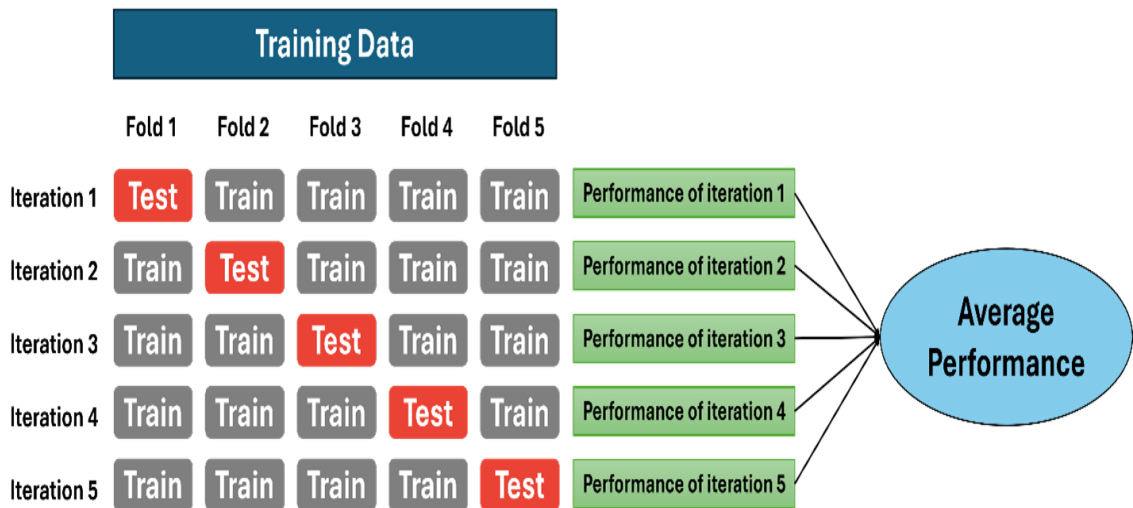


Figure 3.6: Cross-Validation Steps

3.6.1 Cross-Validation Process

The cross-validation process in this study involved several steps, designed to ensure that the models were tested rigorously:

1. **Splitting the Data:** The first stage is splitting the data into n equal subsets, which depends on the adopted value. As for this case, the data set was split into five subsets, equal by size. Further, the first subset is used to test the model, and the other four subsets are used to train the model. This is to be implemented before every subset is used as a test set.
2. **Training and Testing:** For each iteration, the model was trained using four of the five folds. In other words, 80% of the data was used to feed the model, and the other 20% was used for testing purposes. It was done five times; moreover, every time, one of the subsets was chosen as a test set. As a result, the model was tested five times, which differed from all subsets. The difference lies in which subset was used for the test at a specific time, allowing evaluating the performance of the model in multiple scenarios and minimizing the impacts of deviations and errors that one of the subsets could include.
3. **Average Performance Metrics:** After each iteration, I calculated the performance metric of the model, including accuracy, precision, recall, and F1-score. By using the model's performance on various metrics from each iteration, I was able to conclude about the model based on various dimensions of how correctly it classified malicious and benign URLs. In the end, I calculated the average of the performance metrics after the five iterations. This provided a more stable view of the model's performance, compared to relying on a single train-test split.
4. **Mitigating Overfitting:** Cross-validation, especially when used in the form of k -fold, can help in mitigating the problem of overfitting. Since the information about a test set was used only for evaluation, instead of training, the model was unable to learn from it, which also decreased the risk of overfitting. At the same time, the use of multiple training and testing phases provided by the k -fold process, the model was exposed to different fraud test samples at various times, ensuring that the learned patterns will be regarding commonalities and differences between the malicious and benign URLs, as opposed to merely memorizing test sets. In this way, it became more constrained in terms of the kinds of functions it would learn, instead of being guided in its learning towards generalization through various test sets. There was a risk of overfitting in this study as well. However, with the use of cross-validation, I was able to test the model on diverse test sets. This way, the model would be more encouraged to generalize and make generalizable predictions for a wider range of test samples.

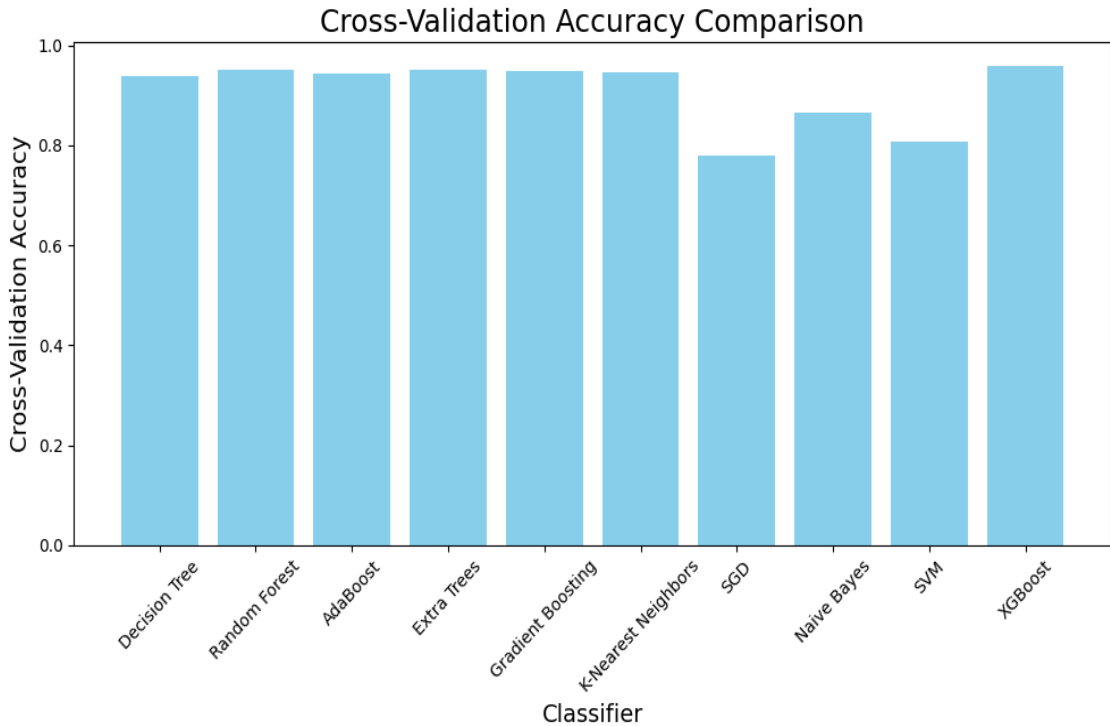


Figure 3.7: Cross-Validation Accuracy Comparison

3.6.2 Performance Metrics Calculated

During each fold of cross-validation, the following performance metrics were calculated:

1. **Accuracy:** The ratio of correctly predicted instances to the total instances.
2. **Precision:** The ratio of true positive predictions to the total positive predictions.
3. **Recall:** The ratio of true positives to the actual positive instances in the data.
4. **F1-Score:** The harmonic mean of precision and recall. It tells us how precise the classifier is and how much it confuses.

By applying 5-fold cross-validation, the system achieved a high performing and fair assessment of the approach in question. The process of 5-fold cross-validation guaranteed that the estimates were relevant to a situation in which the model had to be used to classify malicious URLs. The increased assessment process also facilitated the assurance that the trained model was resilient, maintaining a high level of accuracy while consistently generalizing to out-of-sample data.

Chapter 4

Results and Analysis

4.1 Model Performance

Each ML model's performance was assessed based on key metrics: accuracy, precision, recall, and F1-score. These metrics provided a thorough understanding of how well the models differentiated between benign and malicious URLs.

Table 4.1. Performance Assessment of Machine Learning Techniques

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	96.59%	96.59%	96.59%	96.59%
Random Forest	98.00%	98.01%	98.00%	98.00%
AdaBoost	96.18%	96.19%	96.18%	96.18%
Extra Trees	98.08%	98.10%	98.08%	98.08%
KNN	96.10%	96.10%	96.10%	96.10%
SGD	80.97%	83.02%	80.97%	80.69%
Naive Bayes	89.88%	91.04%	89.88%	89.81%
XGBoost	98.14%	98.36%	97.93%	98.14%

4.1.1 Detailed Observations

1. The best performing model was XGBoost with the highest scores in Accuracy, Precision, Recall and F1-score. Due to its prowess in handling complex feature interactions and overfitting counter-measures, it seems like the most eligible one for final implementation. The gradient boosting technique of XGBoost iteratively improves on prediction errors, making it highly effective in identifying malicious URLs.

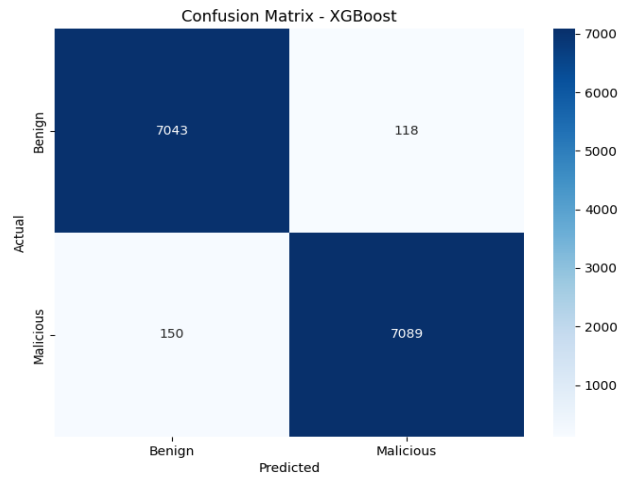


Figure 4.1: XGBoost – Confusion Matrix

2. Random Forest and Extra Trees also performed exceptionally well, with accuracy scores close to that of XGBoost. These models leveraged ensemble learning techniques to reduce variance and improve generalization, particularly in noisy datasets. Moreover size of pkl file of these also was also very high.

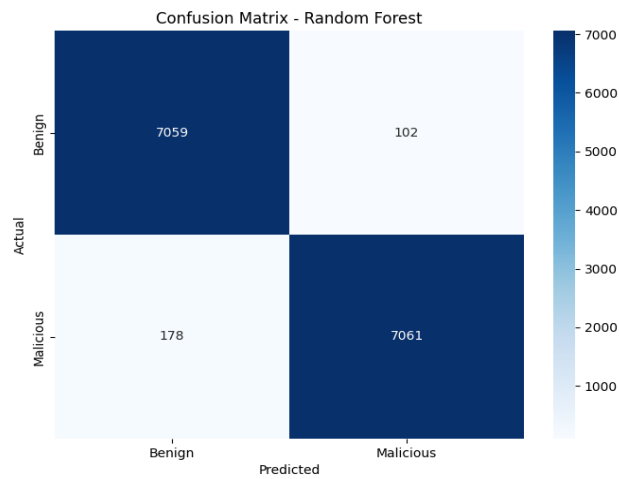


Figure 4.2: Random Forest – Confusion Matrix

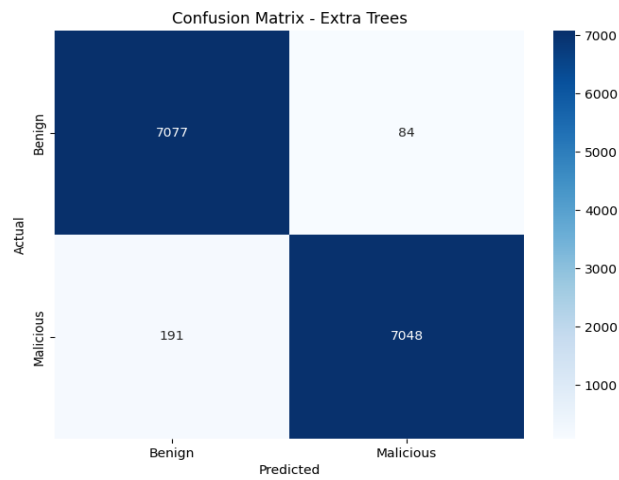


Figure 4.3: Extra Trees – Confusion Matrix

- AdaBoost performed well, achieving 96.18% accuracy. Although its accuracy is slightly more than XGBoost, but it was not selected because it took more execution time than XGboost .

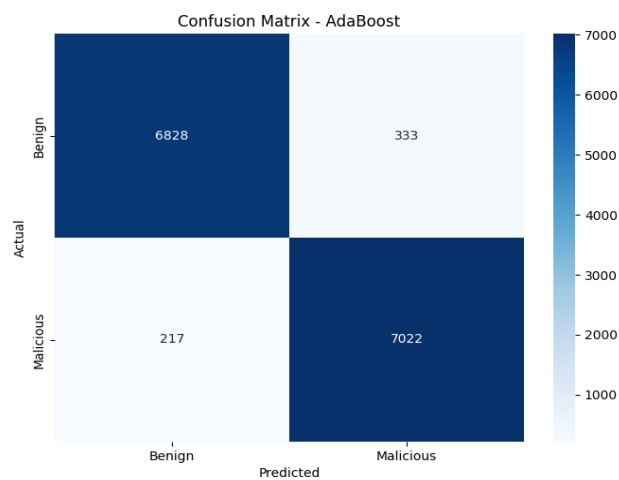


Figure 4.4: AdaBoost – Confusion Matrix

- Decision Tree and KNN showed good performance but were outclassed by ensemble methods like XGBoost. Decision Tree was prone to overfitting, while KNN struggled with computational efficiency, particularly with large datasets.

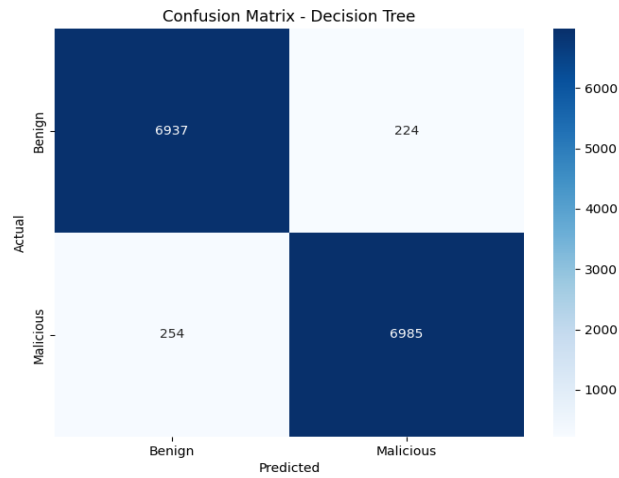


Figure 4.5: Decision Tree – Confusion Matrix

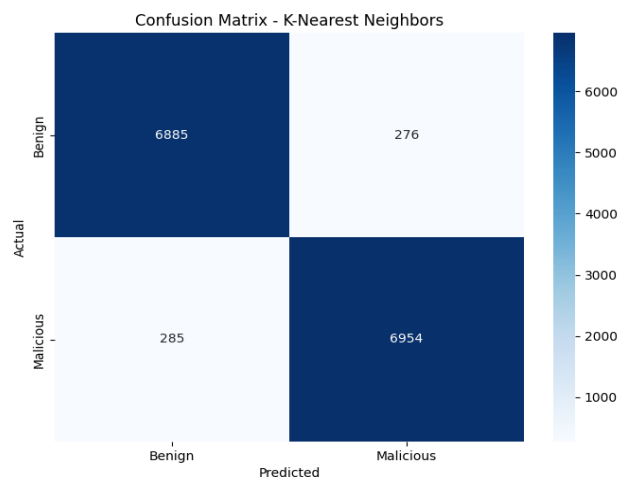


Figure 4.6: KNN – Confusion Matrix

- Naive Bayes and SGD Classifier delivered lower performance, particularly in accuracy and F1-score, due to the limitations of linear models and assumptions of feature independence.

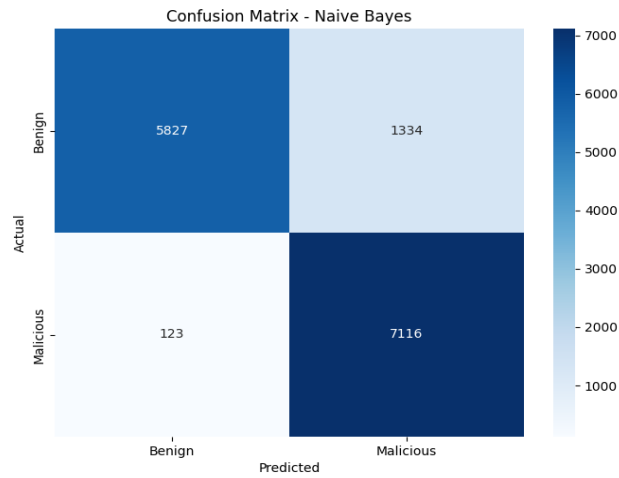


Figure 4.7: Naive Bayes – Confusion Matrix

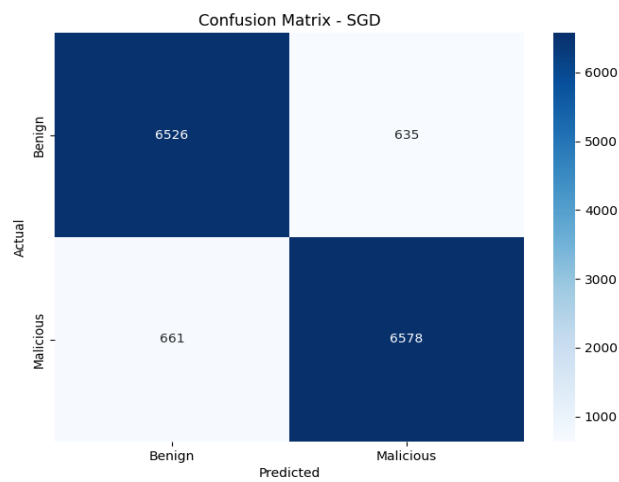


Figure 4.8: SGD – Confusion Matrix

4.2 Feature Importance

An analysis of feature importance was conducted to determine the most influential features in the classification process. XGBoost’s feature importance analysis identified the following top 5 features:

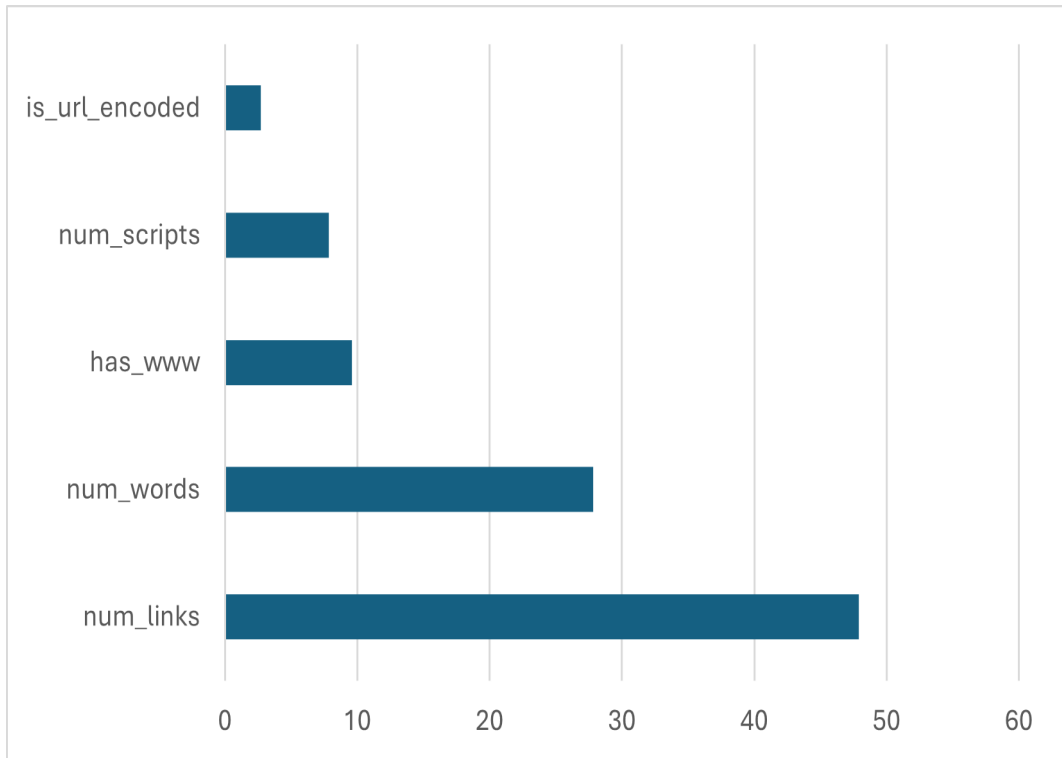


Figure 4.9: Features Importance Score for XGBoost

4.2.1 Key Findings

- **Structural Features:** Features such as number of links and number of words were highly significant, indicating that URLs with a high number of links or words were more likely to be malicious.
- **Security Indicators:** Features like encoded URLs and presence of www were important in distinguishing between benign and malicious URLs. Attackers often manipulate these features to bypass security filters.
- **Content-Based Features:** Number of scripts was identified as another key feature, reflecting how malicious websites often embed suspicious scripts.

4.3 Scalability and Security Evaluation

The system was deployed using a hybrid approach, leveraging both local and serverless environments. The core system, including model training and classification, was deployed locally, while feature extraction was handled via Google Cloud Functions for scalability and cost-efficiency.

4.3.1 Benefits of Using Google Cloud Functions

1. **Efficient Scaling:** Google Cloud Functions dynamically scaled based on the number of URLs being processed. This allowed for seamless scaling during high-traffic periods, ensuring that feature extraction could be handled efficiently without manual intervention.
2. **Security:** Google Cloud Functions operated in isolated environments, providing an additional layer of security during the feature extraction phase. This reduced the potential risk of malicious content affecting the system's core components.
3. **Cost Efficiency:** The pay-per-use model of Google Cloud Functions significantly reduced costs. Resources were only consumed when processing URLs, minimizing idle resource costs.
4. **Reduced Processing Time:** By offloading the feature extraction process to Google Cloud Functions, the average time per URL was reduced for large numbers of URLs, enabling faster processing and improved throughput. It is pertinent to mention that the time for single URL processing is much higher in the hybrid model compared to traditional local processing methods due to cold start latency and network delays. For instance, the time for single URL processing is 2.26 seconds for local processing and 11.09 seconds for the hybrid model (Google Cloud Function). Below is the average feature extraction time:-

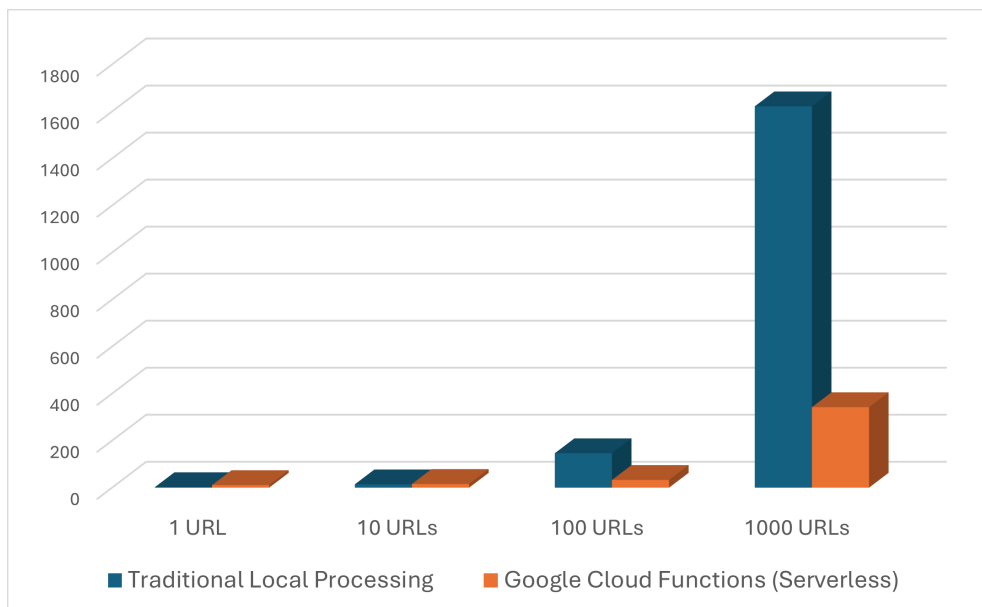


Figure 4.10: Average Feature Extraction Time

Table 4.2. Local vs Google Cloud Function Processing

Metric	Traditional Local Processing	Google Cloud Functions (Serverless)
Scalability	Limited by hardware	Dynamically scales with workload
Resource Management	Manual provisioning	Automatic resource allocation
Cost (1 million URLs)	High due to fixed costs	Lower, pay-as-you-go model
Security	Exposed to local threats	Isolated execution in serverless environment
Maintenance	High, manual updates	Low, managed by cloud provider

By utilizing serverless computing for feature extraction, the system became more scalable and cost-effective, while reducing processing times and maintaining high performance.

4.3.2 Final Model Selection

Based on the performance analysis, XGBoost was selected as the final model for implementation due to its outstanding accuracy, scalability, and ability to handle complex feature interactions. Its gradient boosting technique, which iteratively refines predictions by correcting errors, made it ideal for detecting malicious URLs in real-time scenarios.

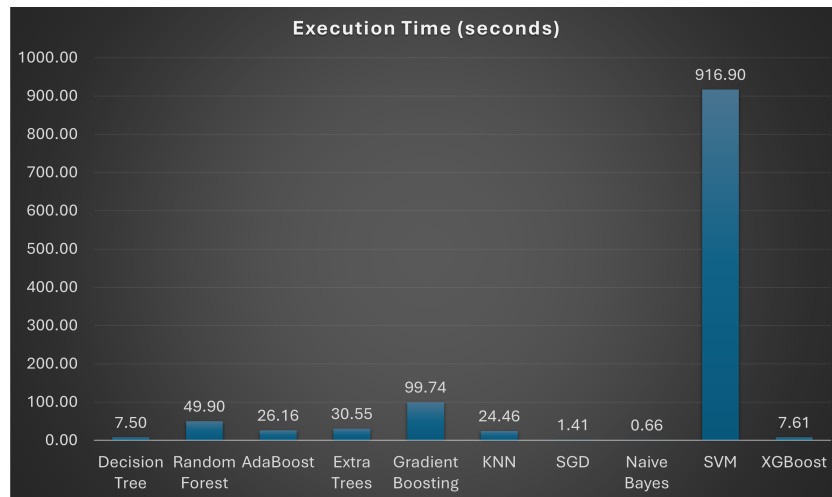


Figure 4.11: Classifiers Execution Time Comparison

4.3.2.1 Why XGBoost Was Chosen

- **Superior Accuracy and F1-Score:** XGBoost consistently outperformed other models in both accuracy and F1-score, ensuring fewer false positives and negatives.
- **Computational Efficiency:** With a time per prediction of 14 milliseconds, XGBoost maintained a good balance between speed and performance, making it suitable for real-time URL classification.
- **Scalability:** XGBoost's ability to scale efficiently with large datasets further cemented its suitability for the final implementation.

By leveraging the superior performance of XGBoost, the system effectively handled real-time malicious URL detection, making it highly suitable for large-scale, real-world applications. The hybrid approach, using both local deployment for model training and Google Cloud Functions for feature extraction, provided an optimal solution for scalability, cost-efficiency, and security.

Chapter 5

Conclusion

5.1 Implications of Findings

The integration of machine learning models with serverless computing provides a robust, scalable solution for detecting malicious URLs. This research highlights several key implications for cybersecurity:

- **High Accuracy and Precision with XGBoost:** The XGBoost model emerged as the top-performing algorithm, achieving the highest accuracy (98.14%) and F1-score. This suggests that XGBoost can effectively detect malicious URLs in real-world applications, reducing false positives and improving detection rates. The model's capacity to handle complex interactions between features, as well as its robustness to overfitting, makes it highly suitable for large-scale cybersecurity systems.
- **Scalability via Google Cloud Functions:** The use of serverless computing, specifically Google Cloud Functions, enabled efficient and scalable feature extraction without the need for manual resource management. This approach allowed the system to dynamically scale with incoming URL traffic, ensuring responsiveness and cost-efficiency. The ability to process high volumes of URLs in real-time demonstrates the system's potential for deployment in environments such as Internet Service Providers (ISPs), security operations centers, and enterprise-level networks where large-scale URL monitoring is necessary.
- **Feature Importance in Malicious URL Detection:** Feature importance analysis revealed critical attributes such as numbers of links, numbers of words, number of scripts, encoded URLs, and presence of www as the most significant predictors of malicious behavior. These findings offer valuable insights for refining detection systems, highlighting that a combination of lexical and content-based

features can significantly improve detection accuracy. Focusing on these key features allows the system to distinguish between benign and malicious URLs more effectively, even in dynamic environments where attackers continuously evolve their methods.

- These findings reinforce that combining machine learning models like XGBoost with a serverless infrastructure can create a highly efficient, scalable, and accurate solution for cybersecurity applications, particularly in detecting and mitigating threats posed by malicious URLs.

5.2 Limitations

Despite the system's strong performance, several limitations must be addressed:

- **Dataset Quality:** The model's performance is directly influenced by the quality and diversity of the dataset. While the dataset used was carefully curated and balanced, real-world URL datasets can be significantly noisier, containing outdated or misclassified entries. These inaccuracies may impact the system's effectiveness when deployed in live environments. To mitigate this, continual data updates and improvements in labeling processes are necessary to maintain optimal model performance.
- **Feature Extraction Complexity and Latency:** Although using Google Cloud Functions for feature extraction significantly reduced processing time, content-based features (e.g., HTML parsing and script analysis) are computationally intensive. During peak traffic periods, this could introduce delays. While serverless computing mitigates resource allocation concerns, further optimization of the feature extraction pipeline is needed to ensure faster and more efficient processing at scale.
- **Cold Start Latency:** One drawback of serverless computing is the "cold start" issue, where functions experience a delay during initial execution after being idle. In real-time detection systems, this latency could impact the system's responsiveness, particularly when handling sporadic or infrequent traffic. Future implementations could explore function warming techniques or optimize invocation patterns to reduce cold start latency and improve real-time performance.

5.3 Future Work

Several areas for future research and development can further enhance the performance and applicability of the system:

- **Real-Time URL Detection with Streaming Technologies:** Implementing real-time URL detection using streaming platforms like Google Cloud Pub/Sub, Apache Kafka, or AWS Kinesis would significantly improve the system’s ability to handle high-traffic environments. This would allow the model to process URLs in real-time, offering faster and more responsive threat detection capabilities.
- **Expanding the Feature Set:** Incorporating additional features, such as network-based indicators (e.g., IP reputation, DNS analysis) or user behavior patterns (e.g., browsing habits or interaction with URLs), could provide a more comprehensive detection framework. These extended features would improve the model’s ability to identify and classify more sophisticated phishing schemes and other malicious activities.
- **Advanced Model Optimization:** Further optimizations could include hyperparameter tuning, model stacking, and even the integration of DL techniques for better detection of complex URL structures. Exploring neural network architectures, such as CNNs or RNNs, could lead to further improvements in handling highly dynamic and evolving threats.
- **Adversarial Attack Mitigation:** As cyber attackers constantly evolve their techniques, future work should focus on improving the system’s robustness against adversarial attacks. Strategies that could be used to make the model more robust against URL manipulation techniques that are designed to bypass detection such as adversarial training or defensive distillation.
- **Deploying in Cloud-Native and Edge Environments:** Future work could explore the deployment of the system in cloud-native architectures or at the network edge. This would bring the detection system closer to end-users, reducing latency and improving threat detection for IoT devices and other decentralized networks.

5.4 Summary of Key Contributions

This research demonstrated the viability of combining advanced machine learning models like XGBoost with serverless computing infrastructure to build an efficient, scalable, and highly accurate system for malicious URL detection. The key contributions of this study include:

- **Development and Evaluation of Machine Learning Models:** The study rigorously developed and evaluated several machine learning models, with XGBoost, Random Forest, and Extra Trees emerging as the top performers. XGBoost, in particular, demonstrated superior accuracy (98.14%) and F1-scores,

showcasing its ability to handle complex feature interactions and detect malicious URLs with high precision and minimal false positives.

- **Utilization of Serverless Computing (Google Cloud Functions):** By leveraging Google Cloud Functions for feature extraction, the system achieved substantial improvements in scalability and reduced processing time, demonstrating the effectiveness of serverless computing for handling high volumes of data without the need for managing underlying infrastructure. The use of cloud functions allowed for real-time URL classification while maintaining cost-efficiency and ensuring dynamic resource allocation based on workload demands.
- **Combining Machine Learning and Serverless Architecture:** The integration of machine learning models with serverless computing resulted in a system that is not only accurate but also scalable, secure, and capable of real-time deployment in diverse environments. The system's dynamic scaling ability, paired with its strong predictive power, makes it an ideal candidate for large-scale URL monitoring in industries such as cybersecurity firms, ISPs, and corporate security networks.
- Provided a clear pathway for future work to enhance real-time detection capabilities, model robustness, and adversarial defense mechanisms.

This research demonstrated that the combination of machine learning and serverless computing can yield a practical, cost-effective, and scalable solution for detecting cyber threats, particularly those posed by malicious URLs.

5.5 Final Thoughts

The findings of this thesis underscore the potential of combining machine learning models with serverless computing infrastructure to address the growing challenges of malicious URL detection. This system, which leverages the strengths of both technologies, offers a powerful tool for real-time identification of phishing attempts, malware distribution, and other online threats. As cyberattacks grow more sophisticated, the importance of maintaining cutting-edge detection mechanisms cannot be overstated. This research lays the groundwork for future developments, highlighting several areas for enhancement:

- **Model Optimization:** Future efforts should focus on refining the machine learning models used, through techniques such as hyperparameter tuning, model ensembling, or even the exploration of deep learning approaches.

- **Real-Time Detection Enhancements:** By integrating streaming technologies such as Google Cloud Pub/Sub, the system could be enhanced to handle high-throughput environments with real-time URL detection capabilities.
- **Expanded Feature Sets:** Additional data sources, such as network-based features and behavioral analytics, could further enhance the system's ability to detect more sophisticated or emerging forms of malicious URLs.
- **Robustness Against Adversarial Attacks:** Implementing defenses against adversarial attacks, where attackers modify URLs to evade detection, will be crucial for improving the system's long-term effectiveness in real-world applications.

With continued refinement and deployment, this system has the potential to become a critical component of organizations' cybersecurity infrastructures, providing robust, scalable, and real-time protection against the ever-evolving landscape of online threats. By addressing the challenges

Bibliography

- [1] *1.6. Nearest neighbors*. <https://scikit-learn.org/stable/modules/neighbors.html>. n.d.
- [2] *1.9. Naive Bayes*. https://scikit-learn.org/stable/modules/naive_bayes.html. n.d.
- [3] Saleem Raja A et al. “Survey on Malicious URL Detection Techniques”. In: *2022 6th International Conference on Trends in Electronics and Informatics (ICOEI)*. 2022, pp. 778–781. DOI: 10.1109/ICOEI53556.2022.9777221.
- [4] *AdaBoostClassifier*. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>. n.d.
- [5] Ravali Attivilli and Angel Arul Jothi. “Serverless Stream-Based Processing for Real Time Credit Card Fraud Detection Using Machine Learning”. In: *2023 IEEE World AI IoT Congress (AIIoT)*. 2023, pp. 0434–0439. DOI: 10.1109/AIIoT58121.2023.10174314.
- [6] Ankit Bhatt, Sachin Sharma, and Shuchi Bhadula. “Security Issues in Serverless Cloud Computing Architectures”. In: *2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT)*. Vol. 5. 2024, pp. 39–43. DOI: 10.1109/IC2PCT60090.2024.10486369.
- [7] *Cloud functions*. <https://cloud.google.com/functions>. n.d.
- [8] Tagba Zoukarneini Difaizi et al. “URL Based Malicious Activity Detection Using Machine Learning”. In: *2023 International Conference on Disruptive Technologies (ICDT)*. 2023, pp. 414–418. DOI: 10.1109/ICDT57929.2023.10150899.
- [9] *ExtraTreesClassifier*. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html>. n.d.
- [10] Bronjon Gogoi, Tasiruddin Ahmed, and Arabinda Dutta. “A Hybrid approach combining blocklists, machine learning and deep learning for detection of malicious URLs”. In: *2022 IEEE India Council International Subsections Conference (INDISCON)*. 2022, pp. 1–6. DOI: 10.1109/INDISCON54605.2022.9862909.

- [11] Muhammed Golec et al. “HealthFaaS: AI-Based Smart Healthcare System for Heart Patients Using Serverless Computing”. In: *IEEE Internet of Things Journal* 10.21 (2023), pp. 18469–18476. DOI: 10.1109/JIOT.2023.3277500.
- [12] *Google Cloud Platform*. <https://console.cloud.google.com/storage/browser?project=remote-features&prefix=&forceOnBucketsSortingFiltering=true>. n.d.
- [13] Hamzamanssor. *Detection malicious URL using ML models*. <https://www.kaggle.com/code/hamzamanssor/detection-malicious-url-using-ml-models>. 2022.
- [14] Xuan Dau Hoang, Dang Le Minh, and Thi Thu Trang Ninh. “A CNN-Based Model for Detecting Malicious URLs”. In: *2023 RIVF International Conference on Computing and Communication Technologies (RIVF)*. 2023, pp. 284–288. DOI: 10.1109/RIVF60135.2023.10471782.
- [15] Shin-Ying Huang et al. “Malicious URL Linkage Analysis and Common Pattern Discovery”. In: *2019 IEEE International Conference on Big Data (Big Data)*. 2019, pp. 3172–3179. DOI: 10.1109/BigData47090.2019.9006145.
- [16] Si Young Jang, Boyan Kostadinov, and Dongman Lee. “Microservice-based Edge Device Architecture for Video Analytics”. In: *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. 2021, pp. 165–177. DOI: 10.1145/3453142.3491283.
- [17] Jawad Khalife, Fatima Tariq Hussain M Nassar, and Maryam Hamad M KH Al Marri. “New Heuristics Method for Malicious URLs Detection Using Machine Learning”. In: *2023 International Symposium on Networks, Computers and Communications (ISNCC)*. 2023, pp. 1–6. DOI: 10.1109/ISNCC58260.2023.10323977.
- [18] Mansi Mehndiratta et al. “Malicious URL: Analysis and Detection using Machine Learning”. In: *2023 10th International Conference on Computing for Sustainable Global Development (INDIACom)*. 2023, pp. 1461–1465.
- [19] Remya R.K. Menon and V Anandhu. “Machine Learning Supported Malicious URL Detection”. In: *2023 4th IEEE Global Conference for Advancement in Technology (GCAT)*. 2023, pp. 1–5. DOI: 10.1109/GCAT59970.2023.10353402.
- [20] *OpenPhish - Phishing Intelligence*. <https://www.openphish.com/>. n.d.
- [21] *PhishTank | Join the fight against phishing*. <https://phishtank.org/>. n.d.
- [22] *RandomForestClassifier*. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>. n.d.
- [23] *SGDClassifier*. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html. n.d.

- [24] Iman Shakeel, Shabana Mehfuz, and Shah Nawaz Ahmad. “Implementing a Serverless Workflow using AWS Step Function”. In: *2023 International Conference on Recent Advances in Electrical, Electronics Digital Healthcare Technologies (REEDCON)*. 2023, pp. 68–73. DOI: 10.1109/REEDCON57544.2023.10150562.
- [25] Thanda Shwe and Masayoshi Aritsugi. “Towards an Edge-Fog-Cloud Serverless Continuum for IoT Data Processing Pipeline”. In: *2024 IEEE International Conference on Big Data and Smart Computing (BigComp)*. 2024, pp. 349–350. DOI: 10.1109/BigComp60711.2024.00063.
- [26] Sikandar. *Malicious-URL-Detection-using-Machine-Learning-and-Serverless-Computing/checkurl.py*. <https://github.com/Sikandar1987/Malicious-URL-Detection-using-Machine-Learning-and-Serverless-Computing/blob/main/checkurl.py>. n.d.
- [27] Sikandar. *Malicious-URL-Detection-using-Machine-Learning-and-Serverless-Computing/Content-based Feature Extraction Code.py*. <https://github.com/Sikandar1987/Malicious-URL-Detection-using-Machine-Learning-and-Serverless-Computing/blob/main/Content-based%20Feature%20Extraction%20Code.py>. n.d.
- [28] Sikandar. *Malicious-URL-Detection-using-Machine-Learning-and-Serverless-Computing/Google Cloud Function Code.py*. <https://github.com/Sikandar1987/Malicious-URL-Detection-using-Machine-Learning-and-Serverless-Computing/blob/main/Google%20Cloud%20Function%20Code.py>. n.d.
- [29] Sikandar. *Malicious-URL-Detection-using-Machine-Learning-and-Serverless-Computing/URL Feature Extraction Code.py*. <https://github.com/Sikandar1987/Malicious-URL-Detection-using-Machine-Learning-and-Serverless-Computing/blob/main/URL%20Feature%20Extraction%20Code.py>. n.d.
- [30] Razvan Stoleriu et al. “Malicious Short URLs Detection Technique”. In: *2023 22nd RoEduNet Conference: Networking in Education and Research (RoEduNet)*. 2023, pp. 1–6. DOI: 10.1109/RoEduNet60162.2023.10274913.
- [31] *VirusTotal*. <https://www.virustotal.com/gui/>. n.d.
- [32] *XGBoost Documentation — xgboost 2.1.1 documentation*. <https://xgboost.readthedocs.io/en/stable/>. n.d.