National College of Ireland

# Boost-AGL Stack: A Scalable and Secure Ensemble Approach for Malicious URL Detection in the Cloud

MSc Research Project
MSc in Cloud Computing

Dasari Lakshmi Narasimha Naga Manikanta
Student ID: x23301295

School of Computing
National College of Ireland

Supervisor: Yasantha Samarawickrama

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

**Student Name:** Dasari. Lakshmi Narasimha Naga Manikanta. …….

**Student ID:** 23301295……..

**Programme:** MSc Cloud Computing **Year:** 2024-2025……..

**Module:** MSc Research Project

**Supervisor:** Yasantha Samarawickrama

**Submission Due Date:** …11-08-2025……

**Project Title:** Boost-AGL Stack: A Scalable and Secure Ensemble Approach for Malicious URL Detection in the Cloud.

**Word Count:** …… **Page Count**…25……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Dasari.Manikanta……

**Date:** 11-08-2025……

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Boost-AGL Stack: A Scalable and Secure Ensemble Approach for Malicious URL Detection in the Cloud

**Abstract**

Malicious URL detection is a critical task in cybersecurity, particularly as phishing attacks increasingly exploit cloud-hosted platforms and services. Traditional approaches often rely on static blacklisting or standalone machine learning models, which struggle with adaptability, scalability, and real-time responsiveness—key requirements in dynamic cloud environments. These methods typically lack deployment automation, consistent performance under varying loads, and integration with modern MLOps workflows. To address these limitations, this study proposes a cloud-native, scalable solution named Boost-AGL Stack, a stacking ensemble model that combines AdaBoost and Gradient Boosting as base learners with Logistic Regression as a meta-classifier. The model is deployed using Docker containers on AWS EC2, with CI/CD automation handled via GitHub Actions, ensuring seamless updates and reliable cloud operation. Experiments conducted on the ISCX-URL-2016 dataset demonstrate that the Boost-AGL Stack outperforms traditional models, achieving the highest accuracy 91%, compared to baseline models like Logistic Regression (90%) and AdaBoost (85%). This confirms the ensemble's robustness and generalization ability. By integrating feature-based classification with cloud deployment strategies, the proposed approach delivers a secure, scalable, and production-ready solution for real-time phishing URL detection. The findings highlight its potential for modern cybersecurity systems operating in cloud ecosystems.

Keywords: Cloud Computing, Malicious URL Detection, Stacking Ensemble, MLOps, AWS Deployment

# 1 Introduction

## 1.1 Aim of the study

The aim of this study is to develop a scalable and secure cloud-native machine learning solution for real-time detection of malicious URLs, with a strong focus on MLOps (Machine Learning Operations) principles. The study proposes a stacking ensemble model—Boost-AGL Stack—that integrates AdaBoost, Gradient Boosting, and Logistic Regression to

improve detection accuracy and model generalization. From a cloud perspective, the objective is to ensure the deployed solution is lightweight, reproducible, and capable of handling dynamic web traffic in real-world scenarios. Using containerization via Docker and deployment on AWS services such as EC2 and ECR, the framework is designed to run efficiently in cloud environments. The integration of MLOps practices, such as automated model packaging, versioning, and continuous delivery through cloud pipelines, ensures that the system can scale horizontally, update models seamlessly, and maintain reliability during high-demand periods. Security is enforced through IAM (Identity and Access Management) roles, and the architecture supports modular updates and real-time API access via Flask. By combining robust machine learning techniques with cloud-native deployment and MLOps practices, the study aims to deliver a high-performance, adaptable, and production-ready system for phishing URL detection, addressing both cybersecurity risks and the operational challenges of deploying ML models in scalable cloud ecosystems.

## 1.2 Research Question

There are some research question for this study:

What is the effectiveness of a stacking ensemble machine learning model in detecting phishing URLs, as measured by accuracy, precision, recall, and F1-score, and how can it be efficiently deployed in the cloud using MLOps practices to ensure scalability, automation, and real-time inference?

## 1.3 Objectives of the Research

The research objectives for this report are:

1. To design and implement a stacking ensemble machine learning model (Boost-AGL Stack) that enhances the accuracy and robustness of phishing URL detection using lexical and domain-based features.
2. To develop a scalable and cloud-native deployment framework using AWS services (EC2, ECR, IAM) and containerization (Docker), ensuring secure and efficient real-time model serving.
3. To apply MLOps practices such as continuous integration, model versioning, and automated deployment pipelines to enable seamless updates and reproducible ML workflows in the cloud environment.

## 1.4 Outline of the Report

This report is structured into seven key sections. Section 1 introduces the research background, objectives, and research questions. Section 2 reviews related work on cloud deployment, containerization, MLOps, and malicious URL detection techniques. Section 3 details the methodology, including dataset description, preprocessing, feature selection, and visualization. Section 4 presents the system architecture and design specifications for the Boost-AGL Stack model. Section 5 explains the implementation steps, including containerization, cloud deployment on AWS, and CI/CD integration. Section 6 evaluates the performance of various models, highlighting the superiority of the proposed stacked ensemble. Finally, Section 7 concludes the study and discusses limitations and future enhancements.

# 2 Related Work

## 2.1 Cloud Computing for Scalable ML Deployment

### 2.1.1 Benefits of Cloud Infrastructure

Cloud computing has changed the process of deployment and scale up of machine learning models especially to real time applications like malicious URL detection (Akinbolaji, 2023). Among the major advantages of cloud infrastructure, it is possible to note the fact that computing capacity and storage are provided on the basis of their demand, thus helping an organization dynamically scale the resources related to the productivity. This elasticity is essential to machine learning workloads where lots of data should be processed, or predictions must be served at scale. Various managed services in such platforms as Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, provide an extensive mechanism of simplifying the deployment of models (Atcha et al., 2024). End-to-end ML pipeline services like EC2 to do virtual computing, S3 to do scalable object storage, ECR to store container images, and EKS to do container orchestration. Automated load balancing, fault tolerance, and autoscaling are also provided by these cloud suppliers and result in a higher performance and more reliable deployed models. Also, cloud platforms can handle infrastructure-as-code, and through it, one may automate deployment infrastructure, monitoring, and version-control. Containerization tools like Docker coupled with the cloud makes deployment reproducible, and transportable, which it inherits in both development and production environments.

### 2.1.2 Challenges in Cloud ML Deployment

However, in spite of all the opportunities provided by cloud platforms, implementing the machine learning models in the cloud is associated with a complex of important difficulties. Among the biggest concerns are the organizational cost of ML workload at scale, one of the most outstanding ones. Pay-as-you-go pricing plans are also costly when large amounts of data or frequent update of models or even in real-time predictions. costs are quite manageable with proper way of allocating resources and selection of compute instances, serverless architectures. The other technical challenge is latency, especially in real-time application, including malicious URL detection, because it may affect user safety (Orozco et al., 2024). This can be in form of latency due to data transmission, cold start serverless platform, or overhead when using container orchestration to distribute data. Also, model versioning in the cloud may be especially complicated in the case where several models versions run together to support A/B testing or incremental rollout (Chitraju Gopal Varma, 2025).

### 2.1.3 Scalable Deployment of Machine Learning Models through MLOps and Containerization

Both of them touch upon the topic of scalable deployment approaches in contemporary software environments, but with a single difference: whereas the former is concerned with enterprise containerization through Docker, Kubernetes, and CI/CD to streamline software development, the latter is about MLOps and the inclusion of CI/CD in machine learning lifecycles. (Omoniyi Babatunde Johnson et al., 2024) mention cost-efficiency and operational resilience as the advantages of ML, and (Garg et al., 2021) cite data pipeline automation and

GitOps deployment strategies as the set-specific issues of ML. Although the approaches are focused on different domains, both of them emphasize cloud-native tools and automation. Its drawbacks however are legacy integration (Omoniyi Babatunde Johnson et al., 2024) and tool fragmentation (Garg et al., 2021), which indicates the necessity to go through standardization and AI-enhanced enhancements to the deployment process.

## 2.2 Malicious URL Detection Techniques in Cloud

(Alkhudair et al., 2020) have suggested a machine learning-based solution to identify malicious URL where online delivery of core services such as banking, healthcare, and social networking pose as ever-increasing cybersecurity threat. This research should improve on web security by detecting malicious URLs automatically before they inflict any damages. Random Forest, Decision Tree, Support Vector Machine (SVM) and K-Nearest Neighbors (KNN) learning algorithms were implemented and tested on two sets of features of the URLs. Of them, the Random Forest algorithm was the most effective with an accuracy of 96% and 95% in the two testing stages, proving its resistance and stability in identifying malicious links. Nevertheless, the paper has been challenged in issues of complexity in critical feature extraction and possibility of overfitting of some of the models. Although the outcomes are promising, one of the limitations is the possible inability to generalize the results to unknown or obscured URLs or even change the attacker tactics, and it is necessary to update the model constantly and replace it with more dynamic and up-to-date sources of information.

The work by (Praba et al., 2024) has suggested a more thorough framework of malicious URLs recognition based on machine learning along with implementation of the Random Forest algorithm and URL parsing mechanisms. This paper concentrates on classifying the URLs as phishing, benign, and defacement according to the key features to be extracted, i.e., domain name, unusual trends, security scores, the existence of URL shortening, and embedded IPs. Based on these characteristics a Random Forest classifier is trained to be able to make the distinction between malicious and safe URLs. The performance of the model is measured with accuracy measures of the training dataset, test dataset, and out-of-bag (OOB) data, and the analysis of important features permits to understand the features with the greatest impact on the classification. The method is effective in the automation of cyber threat such as phishing and defacement thus boosting the security of the web. Nevertheless, the research remains prone to the shortcoming of being incapable of adaptation to fast-changing threat vectors or the inability to respond to massively obfuscated URLs that would not be included as part of the training set.

(Chen et al., 2022) has suggested an optimized version of random forest-based machine learning algorithm to maximize time and accuracy in detecting malicious URLs, besides countering flaws of conventional blacklist-based systems that cannot detect freshly created threats. The objective of the study is the proactive classification of URLs to address the cybercrimes such as phishing, spamming, and drive-by downloads that are extremely dangerous to user privacy and economic security. A number of machine learning models were benchmarked on a self-collected dataset to set the benchmark performance, although the primary goal was to compare and fine-tune the Random Forest classifier into delivering the best performance. The last optimized model had a better classification accuracy and low complexity of computation that was convenient to be used in detecting the URLs in time. Although effective, the strategy could be limited by failure to adapt to new evolution of

attack vectors or the zero-day threats that are not reflected in the training data. In addition, the model is trade-off between performance and efficiency, so further evaluation on real-time and big data would be needed to verify that the model is scalable and invariant to variable environments.

In a time where phishing attacks are increasingly occurring at much faster rates than traditional blacklists and rule-based filters, PhishNot (Alani and Tawfik, 2022) suggests a new mechanism of phishing via the URL prevention of attacks by the use of machine learning. The paper notes how crucial it is to find adaptive and scalable solution to the problem as phishing attacks grow more advanced with millions of new websites being created every month. The data-driven system laid out by the authors relies on a subset of 14 input features, which must be sufficient to provide accurate results and ultimate applicability in the real world. The Random Forest classifier recorded the best performance among the models that have been used with an impressive accuracy of 97.5% with a quick processing time of only 11.5 microseconds per URL once deployed on the cloud. This renders the system efficient and practical in terms of being real time.

A machine learning-based detection framework that uses features extracted out of both malicious and normal network traffic flows to detect anomalous network traffic in a bid to enrich the capabilities of the Intrusion Detection Systems (IDS) has been suggested by (Alshammari and Aldribi, 2021). The research seeks to enhance the accurate detection of the emerging cyber threats that attack the network vulnerabilities using the ISOT-CID repository since it involves the application of features from network traffic circulation and temporal intervals. A new design that relied on the rambling packet length of payload was presented and demonstrated to be effective in training and significantly enhancing the correctness of detection of the model. The algorithm that was used to train this enhanced dataset and then classify it as regular and malicious traffic is machine learning. Among the key issues, which were discussed is how to define and engineer the pertinent features to aptly characterize assorted kinds of attacks in order to have the opportunity of acquiring a robust curve of learning. The results of conducted experiment made it clear that the proposed model can be successfully applied in order to improve the anomaly detection capabilities, which definitely opens the path to their further development. The downsides are however that it is dependent on a certain dataset, thus reducing the possible generalizability.

The approach developed by (Maftoun et al., 2024) states that it utilizes machine learning algorithms to identify malicious websites to cope with the issue of unreliable data input and makes it clear that updating the exhaustive lists of URLs is not feasible. The proposed research study intends to improve on the safety of the web by providing the correct prediction of the websites as such/malicious based on a database of 1,781 entries consisting of 13 features. Some machine learning classifiers such as HGBC, KNN, LR, DT, RF, MLP, LGBM, and SVM were tested after preprocessing procedures which included miss value imputation, normalization as well as class balancing smote. Grid search was used to optimize the models and standard performance measures used to assess the models included accuracy, precision, recall, F1-score, and AUC. Of the models tested, the Hist Gradient Boosting Classifier (HGBC) performed best as a whole, having a stronger tendency to detect, across all the evaluation measures.

Finally, (Kandula et al., 2024) has developed an enhanced cybersecurity mechanism where the mechanism integrates machine-learning-based URLs with a behavioural analysis of URLs as the malicious components to deal with the changes of online threats. In making the choice

of collected data, it is of interest to note that the research addresses the drawback of the machine learning algorithms in that they exig a numerical format as they convert raw, non-numeric URLs to numerical data forms lexical, making them useful in processing with the XGBoost classifier. The accuracy that the classifier attained in identifying malicious URLs was 94.33%. To enhance the CIA (Confidentiality, Integrity, Availability) triad, the mechanism has been incorporated in the system ability to analyze the behavior of URL in controlled setting to identify complex attack vectors and tactics. This two-fold technique does not only maximize the recognition of threats but also allows greater insight into new tactics developed by cyberattackers. Nevertheless, the authors of the study admit that it can be narrowed down with the help of dependence on the correctness of feature transformation applied on the raw fragment of the URL address and to the range of observed behavioral patterns, which cannot cover all the new or the extensively obscured methods of attack. Moreover, the aspect of scalability and real-time deployment needs improvement and it might result in practical implementation in grand scale or in high-speed networks.

Table 1: Summary Table

| **Author(s) & Year** | **Approach / Techniques** | **Dataset / Features** | **Best Algorithm & Accuracy** | **Strengths** | **Limitations** |
|---|---|---|---|---|---|
| (Alkhudair et al., 2020) | ML-based detection using RF, DT, SVM, KNN on URL features | Two sets of URL features | RF: 96% & 95% (two testing stages) | High accuracy, stability in detection | Complex feature extraction, risk of overfitting, poor adaptability to new/obscured URLs |
| (Praba et al., 2024) | RF + URL parsing for phishing, benign, defacement classification | Features: domain name, trends, security scores, shortening , embedded IPs | RF (accuracy not specified in % exact but high in train/test/OOB ) | Effective feature-based classification , explains feature importance | Limited adaptability to fast-changing threats, struggles with heavily obfuscated URLs |
| (Chen et al., 2022) | Optimized RF for time & accuracy, benchmarked with multiple ML models | Self-collected dataset | Optimized RF (better accuracy, low computational cost) | Proactive detection, efficient, better than blacklist systems | May miss zero-day threats, needs big-data real-time validation |
| (Alani and Tawfik, 2022) | 14 URL features, cloud-based RF | Phishing detection | RF: 97.5% accuracy, 11.5 µs processing per URL | High accuracy, real-time, scalable | Relies on fixed set of features, may miss novel attack vectors |
| (Alshammar | ML on | Features | ML model | Enhances | Dataset- |

| | | | | | |
|---|---|---|---|---|---|
| i and Aldribi, 2021) | malicious & normal network traffic flows (ISOT-CID) | from traffic circulation & temporal intervals | (accuracy not specified) | IDS, robust detection | specific, limited generalizability |
| (Maftoun et al., 2024) | Multiple ML classifiers with preprocessing (SMOTE, normalization, imputation) | 1,781 URLs, 13 features | HGBC (best overall) | Balanced evaluation across metrics, robust detection | Dataset size small, may not scale well |
| (Kandula et al., 2024) | XGBoost + behavioral analysis of URLs | Lexical features + behavioral patterns | XGBoost: 94.33% accuracy | Two-fold detection (features + behavior), deeper threat insight | Dependent on feature transformation quality, limited scalability in real-time large-scale networks |

# 3 Research Methodology

## 3.1 Dataset Description

The dataset utilized for this study is sourced from the ISCX-URL-2016 repository, available through the Canadian Institute for Cybersecurity having link: http://cicresearch.ca/CICDataset/ISCX-URL-2016/Dataset/. It is a comprehensive collection of URLs specifically designed for research in malicious website detection. The dataset reflects the increasing threat landscape on the web, where URLs act as a primary vehicle for online criminal activities such as phishing, spam, and malware distribution. Traditional security solutions have largely focused on blacklisting known malicious domains. However, this reactive approach is limited in its ability to counter the rapid emergence of new threats. Many phishing URLs appear temporarily and are often not captured in time, while even trusted domains can be compromised to host malicious defacement URLs.

The ISCX-URL-2016 dataset addresses this issue by providing a labeled collection of different URL types, enabling proactive detection through lexical and domain-level analysis. The full dataset includes five categories: benign, spam, phishing, malware, and defacement URLs. Benign URLs, totaling over 35,000, were harvested from Alexa's top websites using the Heritrix web crawler and validated via VirusTotal to ensure legitimacy. Approximately 12,000 spam URLs were taken from the WEBSPAM-UK2007 dataset, while around 10,000 phishing URLs were collected from OpenPhish, an active repository of phishing threats. Additionally, more than 11,500 malware-related URLs were obtained from DNS-BH, a well-known database of malicious domains.

In our research, we focused specifically on a binary classification task by narrowing the scope to benign and phishing URLs. This simplifies the detection pipeline while maintaining

high relevance to real-world security threats. The dataset supports feature extraction related to lexical structure, domain properties, and content-based signals, allowing the models to detect suspicious patterns and anomalies effectively. This makes the dataset highly suitable for training scalable and secure machine learning models aimed at real-time phishing detection in a cloud-based, containerized environment.

## 3.2 Data Cleaning

The data cleaning stage was carried out in the following way: the first step consisted of data cleaning to make sure the quality and relevance of features to be used in training the model. The initial dataset had a column called the Domain which was considered to be not very useful in the predictive task so it was dropped using the drop() command. The cleaned data maintained the key features in terms of the URL structure, domain attributes and content-indicators. Missing values were checked through dataframe.isna().sum() and denoted that no value was missing in each feature hence completeness of the dataset.

Dataframe.describe() produced descriptive statistics to help with the understanding of distribution and central tendencies of features. The data consists of 2,000 samples, 14 (numeric types) feature columns and one column that is a binary label (phishing or benign URL). The 'https_Domain' and 'Have_IP' features were also fixed with mean of 0 and 1 respectively meaning that they had no variance and continued being ineffective in learning. Pieces of information such as the URL_Depth were also sufficiently different with a minimum and maximum value of 0 and 16, respectively, illustrating a different arrangement of URL paths. Majority of features were binary that made the models interpretable. Label distribution was balanced since it had a mean of 0.5, which means it had equal chances of getting the two classes. This was a clean, well structured dataset which would be effectively visualized, preprocessed and or modeled.

## 3.3 Data Preprocessing

The preprocessing part of data was essential to ensuring the training of machine learning models would be effective. At the beginning stage, the data set was divided into target variables and features. This feature set X was obtained by discarding the 'Label' column, and the target variable Y was given the values of the original column of the datum set corresponding to the 'Label' column, i.e. the output encoding the statement that the URL is phishing (1) or benign (0). In order to have a fair evaluation of the model and preserve the class distribution, the data split was performed between training and testing sets, and stratification was enabled with the ratio of 80:20. To enhance generalization of the data, the train_test_split() was used with the parameters shuffle=True.

After splitting, feature importance analysis has been carried out in order to know the attributes contributing considerably to the task of classification. A model that is an ensemble-based, ExtraTreesClassifier, which is famous when it comes to calculating the relevance of the features, was trained on a training set. The averages of the feature importances were taken per all the trees in the ensemble. The features with the highest influence were distinguished and visualized in form of a bar chart. These were features such as URL_Depth, Prefix/Suffix, Web_Traffic which had a high discriminative capacity. This step did not only identify

important predictors, but also identified insights on how to select the potential features and thus made the data interpretable and efficient in model learning.

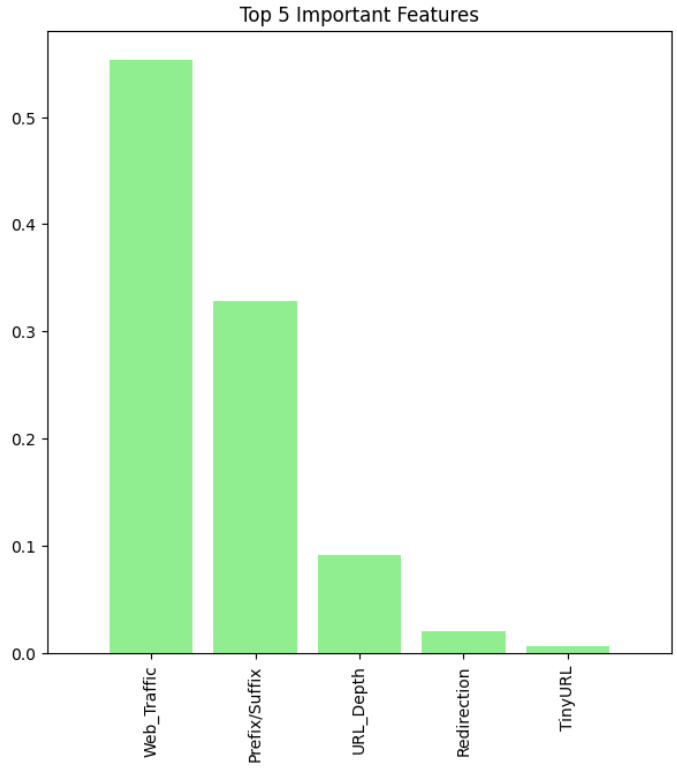


Figure 1: Top 5 Important Features

Figure 1 illustrates a bar chart representing the top five most important features influencing the phishing URL classification model, as identified by the ExtraTreesClassifier. The Web_Traffic feature holds the highest importance score of approximately 0.57, followed by Prefix/Suffix at 0.33. These two features contribute significantly to model decisions. URL_Depth ranks third with a lower importance of around 0.09, while Redirection and TinyURL contribute minimally with scores close to 0.02 and 0.005 respectively. This analysis helps prioritize features during model training and highlights which URL characteristics are most predictive in identifying phishing attempts.

## 3.4 Data Visualization

Figure 2 presents a horizontal bar chart illustrating the distribution of target classes in the dataset, specifically representing benign (label 0) and phishing (label 1) URLs. The chart shows a nearly balanced dataset with 1,000 benign URLs displayed in blue and 1,000 phishing URLs displayed in red. This balance ensures fair model training and prevents bias toward any particular class. A balanced dataset is critical in classification tasks to achieve reliable performance across both classes. The even distribution supports effective learning and evaluation of phishing detection models without skewed outcomes.

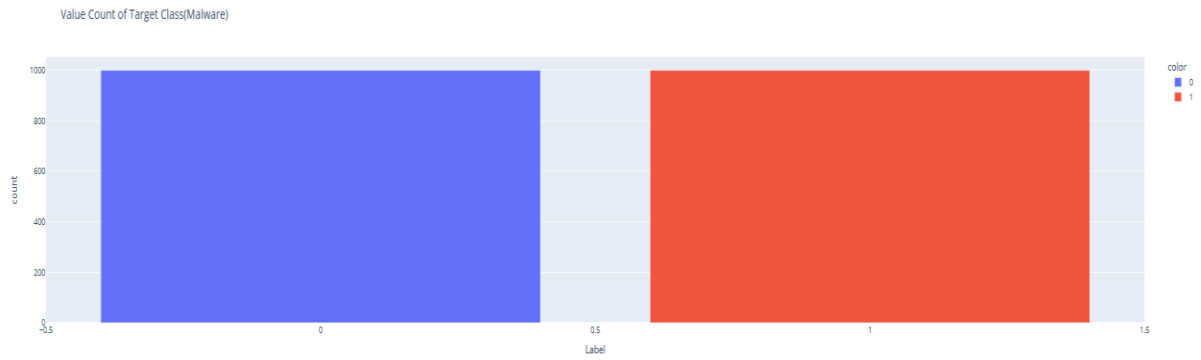


Figure 2: Distribution of target classes

Figure 3 displays a stacked bar chart showing the relationship between the Redirection feature and the target labels in the dataset. The x-axis represents the redirection status—0 (no redirection) and 1 (with redirection)—while the y-axis indicates the count. The majority of URLs, approximately 1,950, fall under redirection value 0, with both phishing (yellow, ~975) and benign (dark blue, ~975) classes almost evenly distributed. Very few samples, roughly 50 in total, exhibit redirection value 1, mostly corresponding to phishing URLs. This figure highlights that most URLs do not use redirection, but its presence may slightly correlate with malicious behavior.

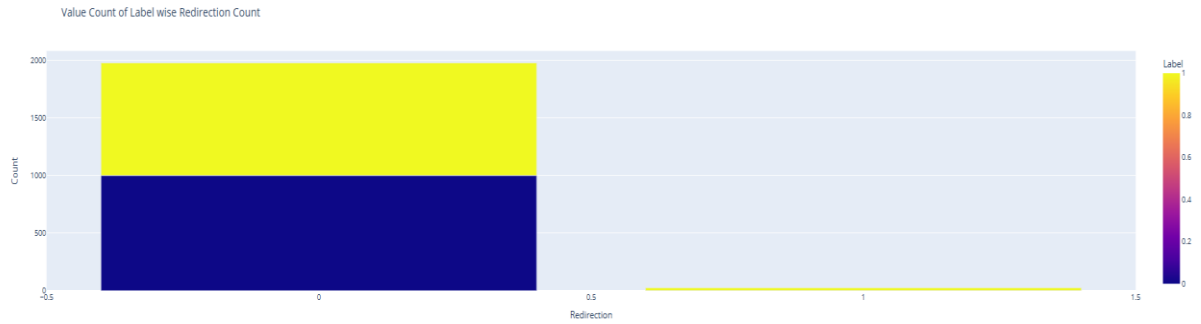


Figure 3: Value count of label wise redirection count

Figure 4 presents a stacked bar chart showing the distribution of phishing and benign URLs across various URL depth levels. The x-axis represents URL_Depth ranging from 0 to 16, while the y-axis shows the count of URLs. A significant concentration is observed at depth 1, with around 750 URLs—majorly phishing (yellow). Depth levels 2 to 5 each have 200–300 samples, with a mix of phishing and benign (shaded purple to dark blue). Beyond depth 6, the counts drop steeply, with very few URLs having depth above 8. This figure suggests that shallow URL paths, particularly depth 1, are more prone to phishing, making URL depth a vital feature for detection.

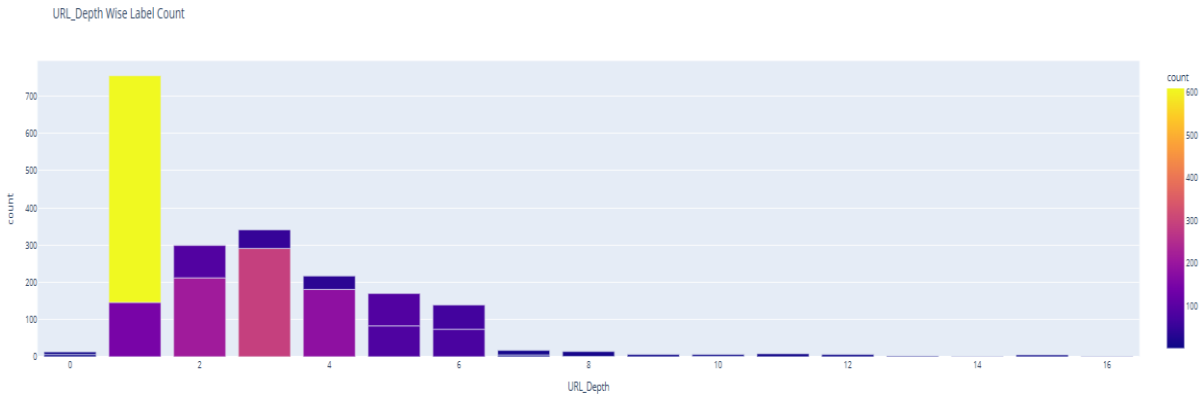


Figure 4: URL_depth wise label count

Figure 5 is a pie chart representing the distribution of URLs across different URL_Depth values in the dataset. The highest proportion of URLs, 37.8%, have a depth of 1, indicating a simple URL structure—often associated with phishing attempts. Depth 2 accounts for 17.1%, followed by depth 3 at 14.9%, depth 4 at 10.8%, and depth 5 at 8.5%. Depth 6 appears with 6.95%, while all other depths from 7 to 14 contribute less than 1% each. This chart highlights that most URLs have shallow paths, reinforcing the importance of URL depth as a distinguishing feature in malicious URL detection.

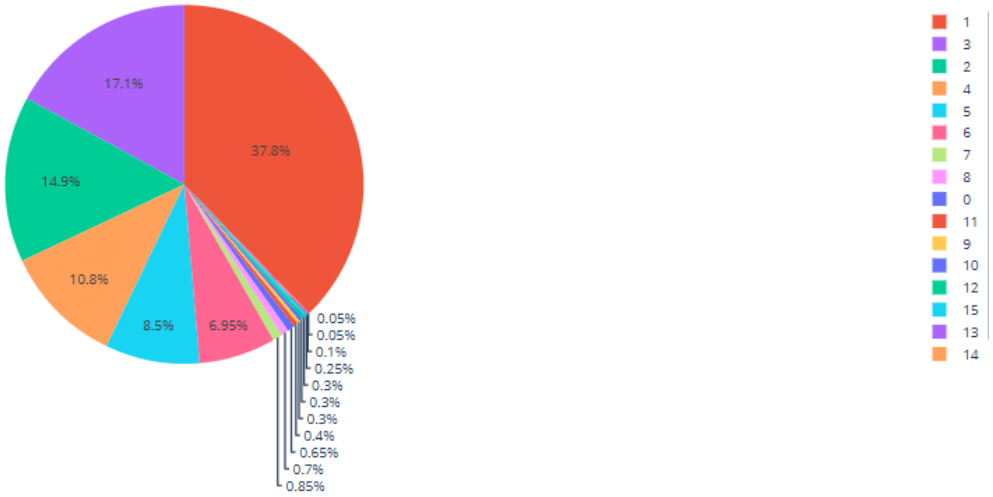


Figure 5: Pie Chart of URL Depth Distribution

Figure 6 shows a heatmap visualizing the Pearson correlation coefficients among key features and the target Label (0 = benign, 1 = phishing). The most significant positive correlation with the label is observed for Web_Traffic (0.80) and Prefix/Suffix (0.71), indicating these features strongly influence phishing detection. URL_Depth and TinyURL have lower correlations of –0.14 and –0.097, respectively. Negative correlation between Redirection and Prefix/Suffix (–0.46) is also notable. Diagonal values are 1.0, showing perfect self-correlation. This heatmap guides feature selection by highlighting attributes with strong predictive power and reducing redundancy in the dataset.

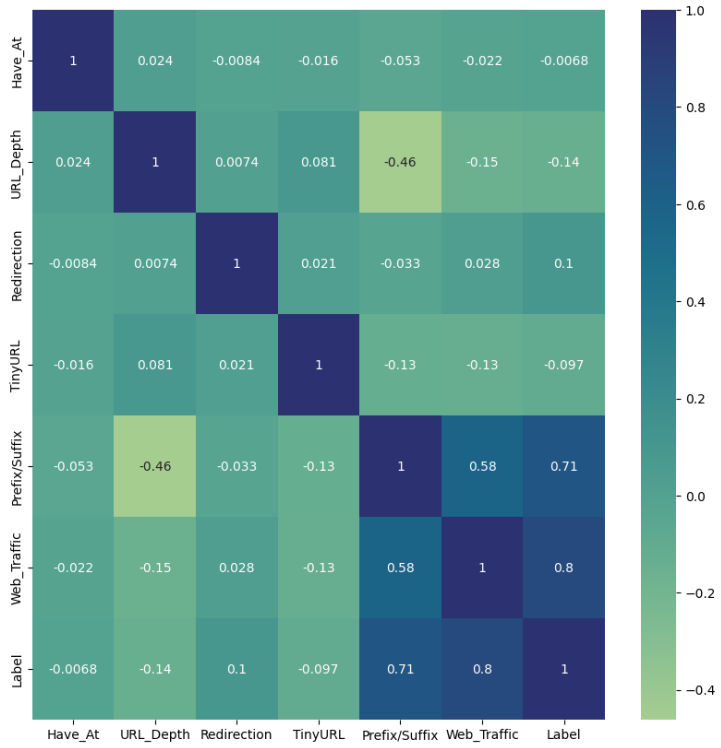


Figure 6: Correlation Matrix

# 4 Design Specification

Figure 7 illustrates the complete workflow and system architecture used in this study for phishing URL detection using the Boost-AGL Stack model. The process begins with importing the required Python libraries and loading the dataset into the system. Following this, data preprocessing is performed to clean, encode, and split the dataset, preparing it for model training. The processed data is then used to train the Boost-AGL Stack, which combines AdaBoost and Gradient Boosting as base learners, with Logistic Regression as the meta-classifier. Once trained and evaluated, the model is embedded into a Flask-based web application to create a lightweight API endpoint.

This application is then containerized using Docker and stored in Amazon ECR (Elastic Container Registry). A CI/CD pipeline is configured using GitHub Actions, which automatically builds the Docker image from the GitHub repository and pushes it to ECR upon updates. The container is deployed on an AWS EC2 instance for hosting. AWS Cloud9 is used for managing deployment scripts and accessing the instance. Once deployed, the Flask API becomes accessible via a public URL from the EC2 instance. This setup ensures a scalable, automated, and secure architecture for real-time malicious URL detection in the cloud environment.

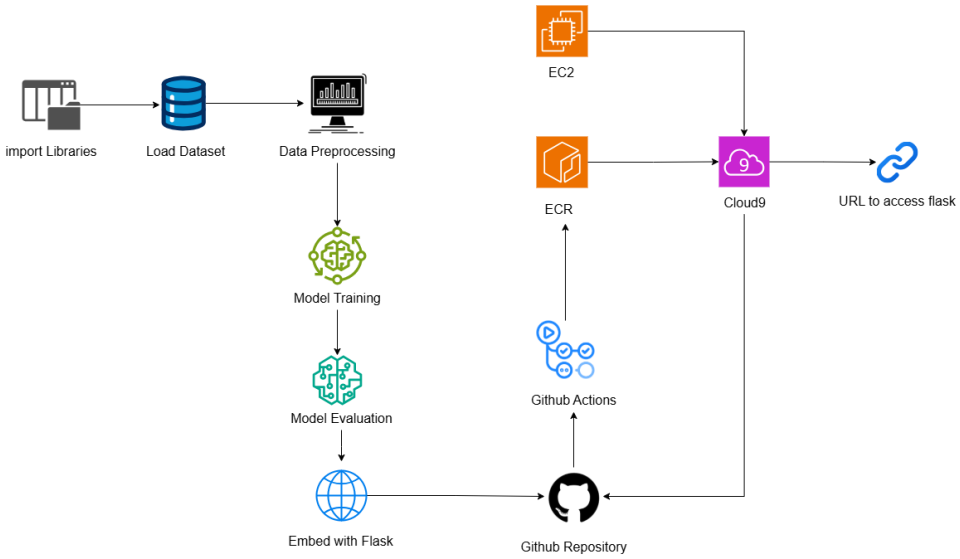


Figure 7: System Architecture Diagram

# 5 Implementation

## 5.1 IAM Role Creation

In the proposed study we use IAM roles to grant access to different AWS services in a controlled manner, without placing the access credentials in the code. It was done by creating a specific IAM role with permission set to a fine-grained granularity that enabled the EC2 instance to safely communicate with services such as Amazon ECR. This method will also increase security and allow automatic access to pull container images. The role can be applied to the EC2 transformed instance, and it will be tightly integrated and authorized, providing the possibility to perform an action as reading data stored in ECR or write logs, without such procedures damaging user credentials or losses of access keys.

## 5.2 Creating ECR Repository

Centralized repository to store, deploy and manage docker images was constructed using Amazon Elastic container registry (ECR). The AWS Management Console is used to create a private ECR repository where authentication AWS EC2 instances access policies are created. This arrangement offers an efficient and safe storage facility of container images, which can be easily paired with the deployment pipeline. With the application of ECR, the newest versions of the model and its runtime environment are always available to EC2 instance and is not dependent on any version and can be deployed consistently with another environment or team.

In figure 8, Amazon Elastic Container Registry (ECR) interface can be seen where the Docker container image of the phishing URL detection model is kept. The repository is generated and is encrypted by AES-256 encryption with mutable tag immutability. ECR is a centralized and scalable container image registry that can be utilized by giving developers clear pushing, storage, and retrieving of container images. This diagram demonstrates how the project uses the ECR in order to host the Docker image, which is subsequently deployed

to an EC2 instance. It guarantees secured and versioned storage of images including support of CI/CD automation and uniform deployment.

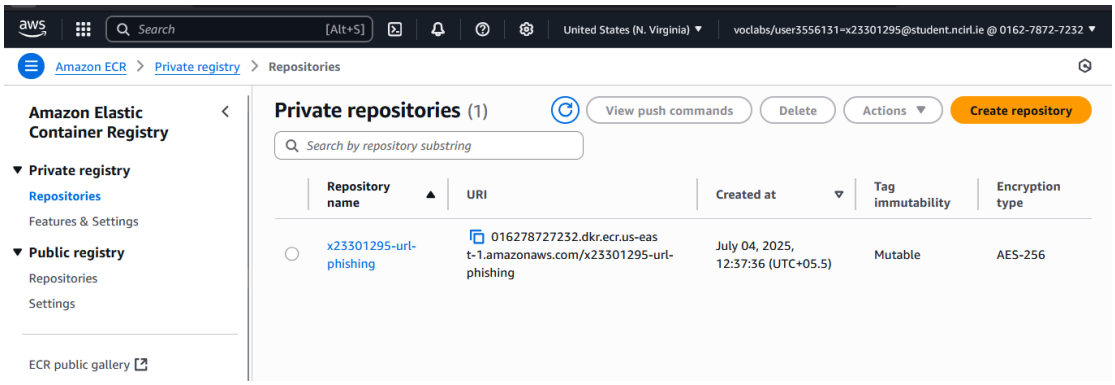


Figure 8: ECR Repository

## 5.3 Push Docker Container to ECR

The first step was to containerize the malicious URL detection Flask application, thus making it portable and deployment consistent. The local image was created and later marked and pushed towards the ECR repository. It involved authorizing the Docker client to ECR with the use of AWS CLI commands then building the Docker image and pushing it safely to ECR. The approach allows version control of application images, ease of deployments and can make the deployed container to be used in the EC2 to be a footprint of the environment in which it is verified locally. This step is an important part of making it possible to deploy a model in the cloud as much as is reproducibly.

## 5.4 AWS Elastic EC2 Instance Creation

Amazon Linux 2 AMI was used to start an EC2 instance to be used as the host environment to run the Docker container. The type of instance was selected according to the compute requirements of the model, and it was assigned with a proper IAM role and security group settings that allow incoming connection to the port 5000 of Flask APIs. After startup, the instance was fitted with Docker and some necessary CLI tools. This arrangement establishes an elastic and re-scalable infrastructure, where to house the containerized ML model, so that real-time predictions are obtainable on API endpoints exposed at the public IP of the EC2.

## 5.5 Installing Required Libraries

Within the Docker container, all necessary libraries and dependencies were installed to support the application's functionality. This includes libraries such as Flask for API creation, pandas and numpy for data manipulation, and scikit-learn for running trained machine learning models. A requirements.txt file was used to manage dependency versions, ensuring consistent installation across different environments. Additionally, Docker was installed and configured on the EC2 instance to pull and run the image from ECR. This step ensures the model has all tools required to execute inference logic efficiently and handle HTTP requests for prediction.

## 5.6 Running the Docker Container on EC2

Once the container image was pulled from ECR to the EC2 instance, it was executed using Docker's run command. The container exposed port 5000, hosting the Flask API that served prediction results. The command used mapped the EC2 instance's public port 5000 to the container's internal port, allowing external access to the service. This enabled real-time inference by sending POST requests to the running container with URL data. Running containers on EC2 allows full control over the host environment while simplifying scaling, monitoring, and debugging during development and deployment stages.

## 5.7 Sending Request to Exposed Endpoint

After deployment, the Flask API hosted within the container became accessible via EC2's public IP and specified port. External clients or test scripts could send POST requests to this endpoint with URL features in JSON format to receive predictions. This setup enables real-time URL classification by invoking the model through HTTP requests. The Flask application deserializes the incoming data, preprocesses it, passes it to the trained model, and returns the prediction (phishing or benign) in a structured format. This mechanism simulates production-like deployment and validates the model's usability in real-world scenarios.

## 5.8 Integrating GitHub Actions with CodePipeline

To automate deployments, GitHub Actions was integrated into the workflow. A CI/CD pipeline was established where code changes in the GitHub repository triggered workflows that built Docker images, authenticated with ECR, and pushed updated images. A .yml workflow file defined build steps and deployment logic, allowing developers to ensure rapid iteration with consistent builds. This automation eliminates manual intervention, speeds up deployment cycles, and ensures that the latest model or application changes are always live. The pipeline ensures that the model serving infrastructure remains up-to-date, efficient, and synchronized with version-controlled code.

Figure 9 displays the GitHub Actions interface, showcasing the successful execution of a CI/CD pipeline defined in deploy.yml. This pipeline automates the process of building and pushing a Docker container image to AWS ECR for the phishing URL detection application. The workflow is divided into three key stages: setup, build-and-push, and notify-success, indicating the initialization, Docker image creation and upload, and post-build notification respectively. Triggered by a code push to the main branch, the process completes in 1 minute and 28 seconds. This setup ensures continuous integration and seamless delivery of application updates to the cloud, improving deployment efficiency and consistency.

**Repository Link:** https://github.com/manidasari27/thesis.git

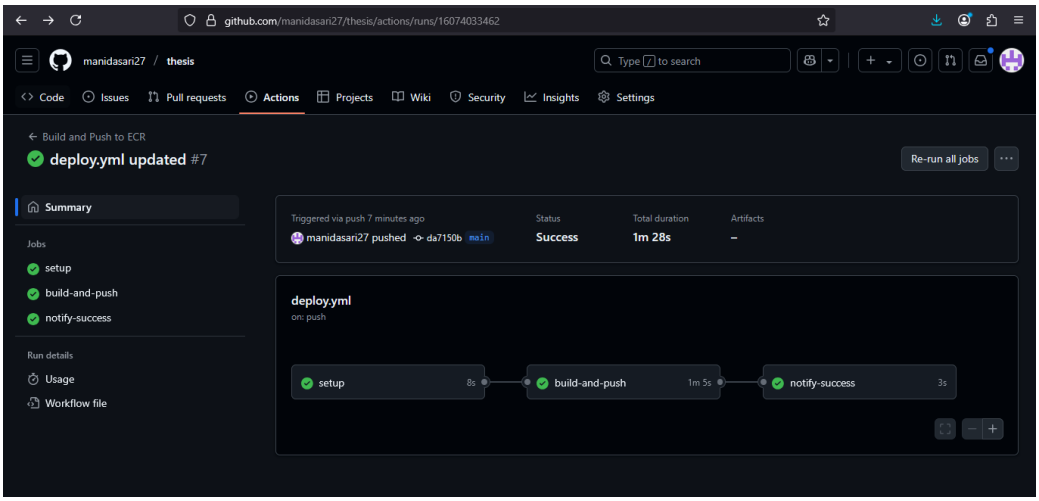


Figure 9: GitHub Actions

# 6 Evaluation

## 6.1 Case Study 1: Logistic Regression

Logistic Regression is a statistical model used for binary classification that estimates the probability of an event occurring by fitting data to a logistic curve. It works well for linearly separable data and is simple, fast, and interpretable.

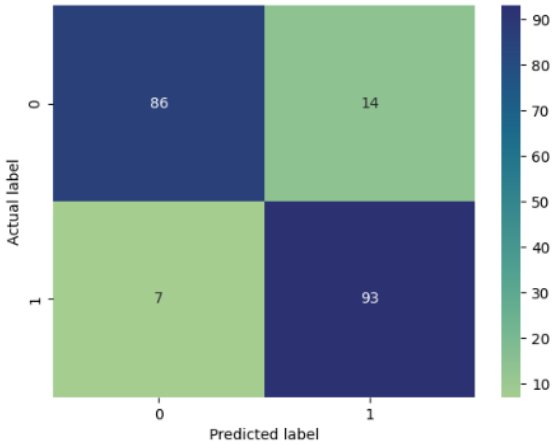


Figure 10: Confusion Matrix for Logistic Regression

Figure 10 illustrates the confusion matrix for the Logistic Regression model used in phishing URL detection. Out of 100 benign URLs (class 0), the model correctly predicted 86 as benign and misclassified 14 as phishing.

## 6.2 Case Study 2: AdaBoost Classifier

AdaBoost, or Adaptive Boosting, combines multiple weak learners, typically decision trees, into a strong classifier by focusing more on misclassified instances in each iteration. It adapts

weights based on previous errors, improving overall performance. In this experiment, AdaBoost achieved an accuracy of 85%. While it had a perfect precision of 1.00 for class 0, its recall was only 0.70, indicating more false negatives. Conversely, class 1 had perfect recall (1.00) but lower precision (0.77), showing imbalance. This suggests that AdaBoost is highly sensitive to one class and may not generalize well across both URL categories.

Figure 11 represents the confusion matrix for the AdaBoost Classifier applied to phishing URL detection. Out of 100 benign URLs (class 0), the model correctly identified 70 but misclassified 30 as phishing, resulting in a high false positive rate. For phishing URLs (class 1), it performed exceptionally well by correctly predicting all 100 instances, with zero false negatives. This leads to a perfect recall of 1.00 for phishing URLs but a lower precision for benign ones. Overall, the model achieved 85% accuracy, highlighting strong sensitivity toward phishing detection but less balance across both classes.

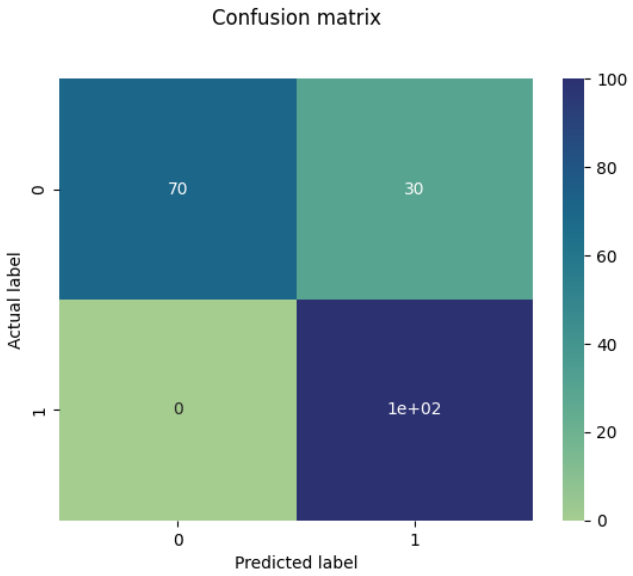


Figure 11: Confusion Matrix for Adaboost

## 6.3 Case Study 3: Gradient Boosting Classifier

Gradient Boosting is an ensemble learning technique that builds models sequentially, where each new model corrects errors made by the previous one. It optimizes a loss function using gradient descent and combines weak learners into a strong predictive model. In this study, Gradient Boosting achieved an accuracy of 90%, with high precision and recall values for both classes. It showed superior recall (0.97) for class 0 and strong precision (0.97) for class 1. The model maintained a good balance, making it effective in identifying both phishing and benign URLs with minimal misclassification.

Figure 12 shows the confusion matrix for the Gradient Boosting Classifier used in the binary classification of URLs. Out of 100 benign URLs (class 0), the model correctly predicted 97 as benign and misclassified only 3 as phishing. For the phishing URLs (class 1), it accurately identified 83 instances but misclassified 17 as benign. The results indicate excellent performance in detecting benign URLs, with a high true positive rate. However, it shows a

slightly higher false negative rate for phishing URLs. Overall, the model achieved 90% accuracy, demonstrating balanced effectiveness and robust performance across both classes.

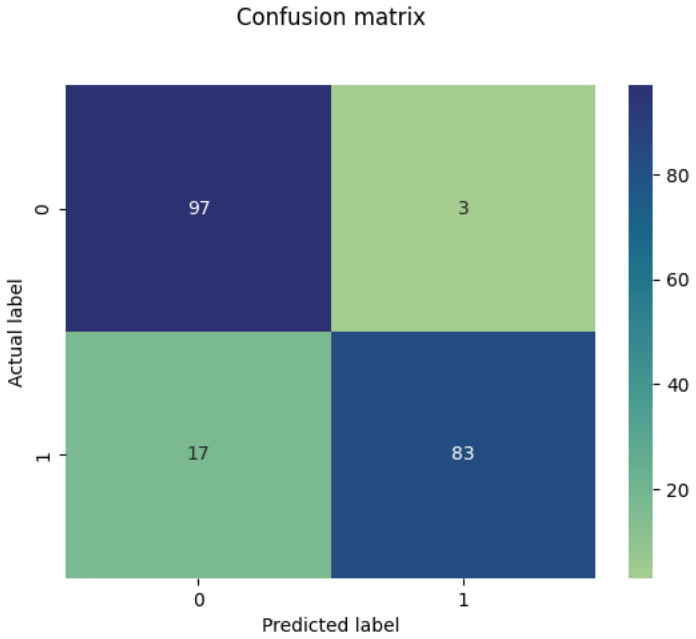


Figure 12: Confusion Matrix Gradient classifying

## 6.4 Case Study 4: Boost-AGL Stack Classifier

The Boost-AGL Stack is a powerful ensemble method that combines the strengths of multiple base models and a meta-classifier to improve prediction performance. In this study, it integrates AdaBoost, Gradient Boosting, and a Logistic Regression meta-classifier. First, the base learners (AdaBoost and Gradient Boosting) are trained on the dataset. Their predictions are then used as input features for the meta-classifier, which learns to make the final decision. This layered approach captures a broader range of patterns and decision boundaries. The Boost-AGL Stack achieved the highest accuracy of 91%, outperforming individual models. It maintained excellent balance across both classes, with precision and recall values consistently around 0.91. This demonstrates its robustness and effectiveness in handling complex classification tasks like malicious URL detection, where combining diverse models helps reduce overfitting and boosts overall generalization.

Figure 13 presents the confusion matrix for the proposed Boost-AGL Stack, the stacking ensemble model combining AdaBoost, Gradient Boosting, and Logistic Regression. Out of 100 benign URLs (class 0), the model correctly classified 96 and misclassified 4 as phishing. For phishing URLs (class 1), it accurately predicted 86, while 14 were incorrectly classified as benign. The matrix reflects a balanced and robust performance with low misclassification rates for both classes. The Boost-AGL Stack achieved the highest accuracy of 91% among all models, confirming its superior generalization and effectiveness in real-time phishing URL detection.

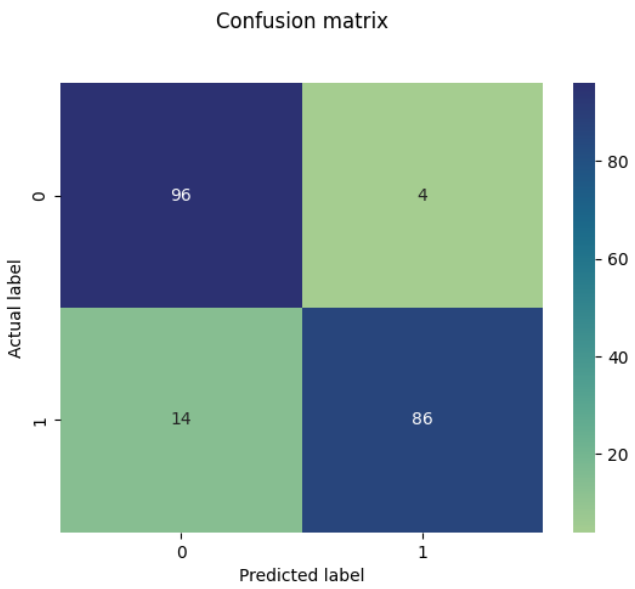


Figure 13: Confusion Matrix for Boost-AGL Stack

The table 2 presents the performance metrics of ML models which evaluated using accuracy, precision, recall, and F1-score. Logistic Regression and Gradient Boosting achieved 90% accuracy, with Gradient Boosting slightly outperforming in precision (0.91). AdaBoost showed competitive results with 85% accuracy. The Boost-AGL Stack emerged as the best-performing model, achieving the highest score across all metrics (0.91), demonstrating balanced precision, recall, and F1-score. These results highlight that ensemble stacking in Boost-AGL enhances predictive performance, making it the most effective approach among the evaluated models.

Table 2: Model Performance Comparison Table

| Model | Accuracy | Precision (Avg) | Recall (Avg) | F1-score (Avg) |
|---|---|---|---|---|
| Logistic Regression | 0.90 | 0.90 | 0.90 | 0.89 |
| AdaBoost | 0.85 | 0.88 | 0.85 | 0.85 |
| Gradient Boosting | 0.90 | 0.91 | 0.90 | 0.90 |
| Boost-AGL Stack (Best) | **0.91** | 0.91 | 0.91 | 0.91 |

## 6.5 Comparative Analysis with Base Work

The proposed Boost-AGL Stack framework surpasses the base work of (Gyimah et al., 2024) both in performance and deployment capabilities as shown in Table 3. While the base paper utilized AutoML with a stacked ensemble on the NSL-KDD dataset for intrusion detection, our study focuses on real-time malicious URL detection using the ISCX-URL-2016 dataset, which is highly relevant in cybersecurity today. Notably, the Boost-AGL Stack achieved a superior accuracy and F1 score of 91%, outperforming (Gyimah et al., 2024)'s 90% accuracy and 89% F1 score. Furthermore, unlike the base work which lacks deployment details, our

approach integrates full-scale MLOps practices including containerization with Docker, cloud deployment on AWS EC2, and CI/CD automation using GitHub Actions. This makes our model not only accurate but also production-ready. By using AdaBoost and Gradient Boosting as base learners and Logistic Regression as the meta-learner, our model maintains strong generalization and low latency, ensuring it meets real-world scalability and security demands.

Table 3: Comparative Analysis

| Criteria | Base Work (Gyimah et al., 2024) | Proposed Work (Boost-AGL Stack – Your Thesis) |
|---|---|---|
| **Dataset Used** | NSL-KDD (Intrusion Detection) | ISCX-URL-2016 (Malicious URL Detection) |
| **Best Accuracy Achieved** | 90% (Stacked Ensemble) | **91% (Boost-AGL Stack)** |
| **Best F1 Score** | 89% | **91% (Boost-AGL Stack)** |
| **Best Precision** | 90% | **91%** |
| **Best Recall** | 89% | **91%** |
| **Base Models Used** | Random Forest, XGBoost, CatBoost, LightGBM | AdaBoost, Gradient Boosting |
| **Meta Classifier** | AutoML (MLJAR Stacked Ensemble) | Logistic Regression |
| **Automation** | AutoML (MLJAR for model selection & tuning) | Manual MLOps pipeline via GitHub Actions |
| **Deployment Strategy** | Not specified | **Containerized deployment using Docker on AWS EC2, ECR** |
| **MLOps Integration** | Minimal | **CI/CD pipeline using GitHub Actions** |
| **Cloud Deployment** | Not implemented | **Yes – Real-time serving on AWS EC2 with Flask API** |

## 6.6 Discussion

The research question focused on evaluating the effectiveness of a stacking ensemble model for phishing URL detection and its efficient cloud deployment using MLOps practices. This was addressed by designing the Boost-AGL Stack, integrating AdaBoost and Gradient Boosting with Logistic Regression as a meta-classifier, trained on the dataset. Rigorous evaluation showed it achieved the highest accuracy (91%) and balanced precision/recall across classes, proving its robustness. Deployment challenges were tackled through Docker containerization, AWS EC2 hosting, and GitHub Actions CI/CD pipelines, enabling scalable, automated, and secure real-time predictions. These results confirm the model's suitability for production-ready cloud cybersecurity applications.

# 7 Conclusion and Future Work

## 7.1 Conclusion

This study shows effective and replicable means of detecting malicious URLs based on a new ensemble model termed as Boost-AGL Stack combines with AdaBoost and Gradient Boosting and Logistic Regression. The study shows that the combination of stacking clustering overcomes the traditional ensemble approach making it effective in real-time phishing in scenarios. Boost-AGL Stack displayed the highest result accuracy of 91% among the tested models and it exceeded the performance of individual models by taking advantage of the strengths of each of the classifiers with multilayered architecture. It was also enhanced by executing the trained model in AWS Cloud through containerization and having automated CI/CD pipes thru GitHub actions. The scalability, portability, and ease in implementing updates are some of the features that are necessary with the deployment of cybersecurity applications. The advanced implementation of machine learning and practical approaches to the cloud-based deployment of the research make the study easily integrated into the production setting. Also, the project focuses on automation, security and efficiency with help of Docker, ECR, EC2 and IAM roles. In general, the study will add a lightweight, high-performing, cloud-ready phishing URL detector that will solve the technical and operational problems in cybersecurity. Boost-AGL Stack is a very powerful stack to meet the present-day requirements in regard to the detection of threats that provide both accuracy and scalability in terms of deployment.

## 7.2 Limitations

Although this model showed high-accuracy and scalability, there are a number of limitations in this study. First, the employed dataset utilized only two classes of the web URLs, phishing ones and benign, which reduces the model capacity to generalization with the domain of other kinds of malicious web URLs like spam, malware, or a defacement. It may have been improved by the addition of more classes to make it more applicable. Second, lexical feature will be helpful in the detection of phishing pattern, but will not retrieve more elaborate obfuscation as well as behavioral characteristic used in the advanced attack. Because of this, URLs that will be designed to bypass lexical analysis may not get detected efficiently. The next limitation is that it is based on the historical data that might not contain the current phishing methods of attackers. Data integration in real-time and in a dynamic manner may enhance adaptability. Regarding the infrastructure, the system was installed into the infrastructure with EC2, Docker, and GitHub Actions, having not applied advanced orchestration tools, such as Kubernetes, which are capable of supporting scaled-up and high-availability deployment. Finally, the model is not tested on dynamic data and does not imply testing or simulation of traffic of a real user, such cases can influence practical performance. These limitations are of significant importance to the long-term efficiency, resiliency, and transferability of the model deployed in a variety of cybersecurity environments.

## 7.3 Future Work

The future works of this study can be done to increase either the amount of data available or the application of deployment ability. The further consideration of such categories of URL addresses as spam, malware, and defacement would allow using a multi-class approach to processing the training set and improving the model performance in real-life scenarios. The next potential avenue is the unification of behavioral and contextual characteristics, like domain age, hosting behaviour and use of JavaScript, which would be useful in scouting more elegant, or even backdoor, phishing tactics. To enhance flexibility, future research is

expected to delve into continuous learning, via running a stream of data around the model, which helps the model keep up with new phishing patterns. On the deployment side, deployment of the Boost-AGL Stack in a Kubernetes or serverless backend (e.g. AWS Lambda) would benefit scalability, resiliency and cost-efficiency. It would also facilitate dynamically managing loads that are extremely important use cases of large-scale enterprises. Besides, we can add an ability to observe and improve the monitors like AWS CloudWatch or Prometheus to trace their security and scale automatically. Future work might also question explainable AI (XAI) methods to allow models to make predictions that are more comprehensible, which would be essential to cybersecurity analysts. Lastly, engaging the industry stakeholders to answer the possible ways of testing the solution in live networks would confirm the practical potential of the solution and determine any additional improvements before the production-level implementation.

# References


AKINBOLAJI, T.J., 2023. Advanced Integration of Artificial Intelligence and Machine Learning for Real-Time Threat Detection in Cloud Computing Environments. https://doi.org/10.5281/ZENODO.13963676

Alani, M.M., Tawfik, H., 2022. PhishNot: A Cloud-Based Machine-Learning Approach to Phishing URL Detection. Comput. Netw. 218, 109407. https://doi.org/10.1016/j.comnet.2022.109407

Alkhudair, F., Alassaf, M., Ullah Khan, R., Alfarraj, S., 2020. Detecting Malicious URL, in: 2020 International Conference on Computing and Information Technology (ICCIT-1441). Presented at the 2020 International Conference on Computing and Information Technology (ICCIT-1441), IEEE, Tabuk, Saudi Arabia, pp. 1–5. https://doi.org/10.1109/ICCIT-144147971.2020.9213792

Alshammari, A., Aldribi, A., 2021. Apply machine learning techniques to detect malicious network traffic in cloud computing. J. Big Data 8, 90. https://doi.org/10.1186/s40537-021-00475-1

Atcha, N.M., Jagannadha Rao, D.B., Polepally, V., 2024. A Contemplate Study on Prominent Cloud Service Providers: Amazon Web Service (AWS), Microsoft Azure, and Google Cloud Platform (GCP), in: Bhateja, V., Lin, H., Simic, M., Attique Khan, M., Garg, H. (Eds.), Cyber Security and Intelligent Systems, Lecture Notes in Networks and Systems. Springer Nature Singapore, Singapore, pp. 223–233. https://doi.org/10.1007/978-981-97-4892-1_19

Chen, J., Hu, Z., Qian, Z., 2022. Research on Malicious URL Detection Based on Random Forest, in: 2022 14th International Conference on Computer Research and Development (ICCRD). Presented at the 2022 14th International Conference on Computer Research and Development (ICCRD), IEEE, Shenzhen, China, pp. 30–36. https://doi.org/10.1109/ICCRD54409.2022.9730451

Chitraju Gopal Varma, S., 2025. End-to-End ML Operations (MLOps): Enhancing Model Reliability and Performance at Scale. https://doi.org/10.2139/ssrn.5226793

Garg, Satvik, Pundir, P., Rathee, G., Gupta, P.K., Garg, Somya, Ahlawat, S., 2021. On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps, in: 2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE). Presented at the 2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge

Engineering (AIKE), IEEE, Laguna Hills, CA, USA, pp. 25–28. https://doi.org/10.1109/AIKE52691.2021.00010

Gyimah, N.K., Mwakalonge, J., Comert, G., Siuhi, S., Akinie, R., Sulle, M., Ruganuza, D., Izison, B., Mukwaya, A., 2024. An AutoML-based approach for Network Intrusion Detection. https://doi.org/10.48550/ARXIV.2411.15920

Kandula, A.R., Phaneendra Kumar, K., Prakash, A.B., Sanjay Kumar, K., 2024. Malicious URL Analysis using Machine Learning Algorithm, in: 2024 8th International Conference on Electronics, Communication and Aerospace Technology (ICECA). Presented at the 2024 8th International Conference on Electronics, Communication and Aerospace Technology (ICECA), IEEE, Coimbatore, India, pp. 1160–1167. https://doi.org/10.1109/ICECA63461.2024.10800903

Maftoun, M., Shadkam, N., Komamardakhi, S.S.S., Mansor, Z., Joloudari, J.H., 2024. Malicious URL Detection using optimized Hist Gradient Boosting Classifier based on grid search method. https://doi.org/10.48550/ARXIV.2406.10286

Omoniyi Babatunde Johnson, Zein Samira, Emmanuel Cadet, Olajide Soji Osundare, Harrison Oke Ekpobimi, 2024. Creating a scalable containerization model for enhanced software engineering in enterprise environments. Glob. J. Eng. Technol. Adv. 21, 139–150. https://doi.org/10.30574/gjeta.2024.21.2.0220

Orozco, D., Quesada, L., Ramirez-Benavides, K., Lara, A., 2024. Real-time malicious URL detection, in: 2024 IEEE Latin-American Conference on Communications (LATINCOM). Presented at the 2024 IEEE Latin-American Conference on Communications (LATINCOM), IEEE, Medellin, Colombia, pp. 1–7. https://doi.org/10.1109/LATINCOM62985.2024.10770685

Praba, M.S.B., Duddukunta, K.R., Bezawada, V.S., Addanki, S.V., 2024. Enhancing Web Security through Machine Learning: A Random Forest Approach to Malicious URL Detection, in: 2024 4th Asian Conference on Innovation in Technology (ASIANCON). Presented at the 2024 4th Asian Conference on Innovation in Technology (ASIANCON), IEEE, Pimari Chinchwad, India, pp. 1–6. https://doi.org/10.1109/ASIANCON62057.2024.10837992