National College of Ireland

Development and Performance Evaluation of a Dockerized Flask Application for Phishing URL Detection Across AWS and Azure

MSc Research Project
MSc in Cloud Computing

Yamini Murugan
Student ID: x23166401

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Yamini Murugan
Student ID:	23166401
Programme:	MSc in Cloud Computing **Year:** 1
Module:	MSc Research Project
Supervisor:	*Shreyas Setlur Arun*
Submission Due Date:	12/12/2024
Project Title:	Development and Performance Evaluation of a Dockerized Flask Application for Phishing URL Detection Across AWS and Azure
Word Count:	8520 **Page Count** 22

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: Yamini Murugan

Date: 12/12/2024

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):	

Development and Performance Evaluation of a Dockerized Flask Application for Phishing URL Detection Across AWS and Azure

Yamini Murugan
X23166401

Abstract

Businesses today predominantly rely on their online presence and are migrating toward cloud solutions for a cost-efficient pay-as-you-go, scalable, and improved reliability model. However, increased dependence on online platforms has exposed organizations and users to various forms of cyber threats, particularly phishing. Phishing, entices users into making monetary transactions and leak their sensitive information, leading to financial loss and data breaches. To combat this problem, this research proposes a phishing URL detection system utilizing Deep Learning models while leveraging the benefits of cloud technologies to ensure high availability and minimal latency. This Phishing URL detection system utilizes Kaggle dataset for benign and phishing URLs, from which 19 features and 1 labelling feature are extracted to build the final dataset for training the DL models. After feature extraction and preprocessing, CNN, LSTM, and BiLSTM models were built to classify the URLs. The BiLSTM model achieving the highest accuracy of 85% was chosen to build the flask app. Furthermore, the application was made portable and easier to deploy by being containerized using Docker and deployed on AWS Elastic Beanstalk and Azure Container Apps. For load testing, Locust was used to generate the traffic with 2000 concurrent users simulated to join at a rate of five users/second. Finally, the performance was documented using AWS CloudWatch and Azure Monitor, and the results were used to evaluate how both cloud platforms performed under various circumstances, benchmarking the CPU usage, network traffic, latency and load average, to compare the strengths and weaknesses of each platform.

1 Introduction

As many organisations learn the endless benefits offered by cloud computing, businesses are seen migrating their operations from on premise to cloud environments to streamline their business processes and achieve operational efficiency. The growing demand for cloud solutions to delivers optimal performance outcomes are at an all-time high, pushing major cloud service providers such AWS and Azure to emulate one another to achieve advanced performance outcomes offering strategic resource allocation, improved availability and minimal latency. Among various challenged faced to achieve cost efficient and scalable solutions, one problem statement under discussion here is to address a critical cyber security threat in the form of phishing, using deep learning models and docker, utilizing cloud services.

One of the most common threats in the current world, which acquires a huge number of victims, is phishing attack that, mainly, aims to steal login credentials and financial information from different users (Alabdan, 2020). With consumers conducting business over the internet, using emails or messenger services for their day-to-day activities, the methods of phishing got more complex, and it is almost impossible for an average user to distinguish between a genuine message and a fake one (Alkhalil Z, 2021). The phosphorylation websites that exist in the social network today are utilized in fighting this prevalent issue by implementing different technologies and algorithms to detect phishing attacks. Cognitive in nature, these platforms use machine learning, natural language processing and heuristic analysis to review the authenticity of URLs and emails depending on URL construction, email header and content features. By using real-time alert and detailed report, the above detection systems improve the online safety, as users are capable of making a decision whether to engage with a particular site based on the available information on the internet. In addition, with the progress of cybercriminals, the problem of identifying phishing has become more important than ever (Adam Kavon Ghazi-Tehrani, 2022). These websites do not only act as a part of a proactive defense, but also help to spread awareness of phishing threats, support users in recognizing many suspicious activities, and generally create a more secure network. The creation of the phishing detection website is therefore critical (N. Q. Do, 2022) in realization of measures pertaining to the risks of online engagements, to safeguard users from falling victims to identity theft and contractual fraud with an associated goal of safe use of the internet and general minimized cases of financial loss. This research is very useful to detect anomalies with the involvement of deep learning neural network models that helps to classify the aforementioned complex URL patterns, resulting in accurate detection of phishing. A method for identifying phishing websites and URLs using machine learning models has been proposed by (Mohammad, 2013). The approach focuses on extracting critical features from the URLs to improve identification accuracy. These features include the URL structure, hyperlink behaviour, website traffic metrics, and domain information. The characteristics of the URLs are classified and composed into a feature extracted dataset to train the ML models. This method enhances existing phishing URL detection tools, increases the accuracy of the classification process, and ultimately helps protect users from online threats, forming the base for this research.

In this research, a web application utilizing the flask framework is created with underlining deep learning model, BiLSTM to detect the phishing URL. The aim of this research is to analyse and apply the best practices for deploying the Flask app on Docker containers on AWS and Azure cloud services. Specifically, utilising Docker's containerization technology, the study aims to guarantee that the application is portable, uniform, and significantly expressible in various cloud spaces. When implementing the Flask application on AWS, the emphasis has been put on using Elastic Beanstalk to increase manageability in addition to establishing a CI/CD using code pipeline to interface with GitHub. This approach not only improves the simplicity of deployment but also manages the regular updates and maintenance processes. Likewise, when deploying on Azure, the Azure Container registry has been configured to Azure container apps, guaranteeing that application will not struggle with different load. Locust is used to put the application to test by generating heavy load, consequently monitoring the metrics with AWS Cloud watch and Azure monitor to identify the bottlenecks and record the maximum performance across both the clouds. The purpose of this comparative study of deployment strategies is to identify benefits and risks related to utilised cloud providers and support the understanding of how cloud infrastructures enhance the overall performance of web applications. Learning the performance outcome is

extremely essential for organizations to identifying the services that are critical and scale their application accordingly, the metrics collected will help businesses to tackle problems related to the health of the application, resource allocation and in case of any sudden failures, enables them to quickly adapt and look for recovery options across multiple zones. Lastly, this study will help developers and organizations interested in utilizing cloud technologies and containerization in their application deployment, for better efficiency and security in today's rapidly changing technology environment.

1.1 Research Question

How does the performance of a Dockerized Phishing URL detection Flask application differ when deployed across cloud platforms like AWS and Microsoft Azure?

1.2 Objectives

The core research objectives include,

1. Identifying the phishing URLs using DL models such as CNN Convolutional Neural Network, LSTM - Long Short-Term Memory and BILSTM - Bidirectional LSTM, after feature extraction and comparing their performance using accuracy score and classification report to visualizing the same using confusion matrix and plot graphs.
2. Create a Flask Application for the model with highest accuracy (BiLSTM) to detect the phishing URLs.
3. Employ Docker containers to ease the deployment and at the same time, place emphasis on issues concerning compatibility, use of resources as well as, the manner in which scaling can be made easy within AWS or Azure services.
4. Outline and implement a CI/CD system to update and deploy the new models, utilizing the AWS CodePipeline and Azure container registry for deployment.
5. Test for load using Locust and monitor the performance metrics of the deployed phishing URL detection application using AWS CloudWatch and Azure monitor.

The below table outlines how subsequent sections of this report are structured.

Section	Structure Description
1. Introduction	Introduces the background, Motive, Research Question and Objectives of the research topic.
2. Related Work	Critically reviews and evaluates multiple literature study.
3. Research Methodology	Outlines the entire process of training the DL Models, Dockerization and deployment.
4. Design Specification	Discusses the Architecture and Tools used.
5. Implementation	Maps the detailed flow of end-to-end implementation for the deployment process on AWS and Azure.
6. Results & Evaluation	Captures the performance testing using Locust and AWS CloudWatch and Azure Monitor metrics for evaluating the end results.
7. Conclusion & Future Work	Summarizes key takeaways, discusses limitations, and potential directions for future research.

Table 1: Report Structure

2 Related Work

For this research, multiple literature papers have been critically scrutinized pertaining to the Deep learning models utilized to accurately categorize the URLs and the process adopted to deploy the Docker Image on multiple cloud environments to extract the performance metrics. These papers lay the foundation and act as the motivation to conduct this study.

2.1 Traditional Approaches for Phishing Detection

Phishing detection methods that have been previously and traditionally employed include blacklisting, content-based filtering and heuristics methods. Blacklisting implies usage of a set of known phishing URLs, meaning that all site access is filtered out of dangerous ones. Content based filtering is based on scanning the body of an emailed message and the URLs within it trying to identify suspicious patterns and using Naive Bayes' classification models to distinguish between genuine and phishing attempts. Heuristic-based detection aims at the exploitation of specific keywords and patterns of the URL in the attempt to predict frequent phishing cases. However, as the evaluation of these models suggests these methods are capable of detecting phishing attacks, there are still current issues in the areas as adaptability and accuracy that underscore the need to consistently update and diffuse the current traditional approaches to contend current threats in an efficient way.

As the fight against phishing continues three large scale studies address various aspects of phishing prediction and prevention. The first study is provided by (Simon Bell, 2020) who compares the efficiency of phishing blacklists including Google Safe Browsing (GSB), OpenPhish (OP) and PhishTank (PT) as well as discusses the shortcomings of existing systems, for instance, the temporary nature of phishing URLs and absence of a one-time-only URL policy. With a relatively greater size of 1.6M URLs on average than OP's 3861 and PT's 12433, GSB still exhibited more impressive detection performance, detecting over ninety percent of repeats before OP. The second study described by (I. Kim, 2023) who presents the Privacy-preserving Content-based Spam Filter (PCSF), which shows enhanced challenges on the nature of privacy preservation during spam filtering and the danger of disclosing email content or loading spam into the users' memory. This paper shows that by employing pre-validation prior to email reading and Naive Bayes spam filter in PCSF, security is increased with added ESP spam filtering capabilities without violating user privacy. The third study did include a survey on heuristic-based strategies which is presented by (Carlo Marcelo Revoredo da Silva, 2020) for anticipation of phishing by URL analysis where it concerned only with static attractions such as keywords and patterns. This analysis states that, as research moves ahead some features may still be less useful and highlights the need to adapt with the changing phishing strategies. The qualitative analysis of this research work identifies related features and insights into feature relevance and relationships, thus suggesting that some of the features being repeatedly identified in the phishing URLs could be harnessed for better detection. Altogether, these works raise the necessity of sound, versatile, and private anti-phishing solutions, showing directions for subsequent research to expand the effectiveness of identification and counteraction methods.

2.2 Machine Learning Approaches for Phishing Detection

Several studies have adopted different machine learning strategies to detect phishing and show good results in deciphering malicious activeness. Some of the popular model used are Support Vector Machine reflected by SVM, Random Forest and Classification Trees and the

said models have been proven to have high accuracy as it pertains to classification between the two. Moreover, incorporation of nature-inspired optimization algorithms to improve SVM performance is also applied. In general, these papers show that machine learning methods can be useful in dealing with phishig threats. Multiple papers were researched on various Machine learning models, optimization techniques and methods to combat Phishing more effectually. (Diki Wahyudi, 2022) presented a phishing detection system based on machine learning for detecting and addressing the menace of cyber-criminal activities. The process included using a training set consisting of 11, 055 Web sites identified as 'legitimate' or 'phishing' to successfully classify and identify phishing attempts. The proposed model was assessed using a 10 times cross-validation in order to have an indication of how the model will perform on other samples of the data. Three algorithms namely, Support Vector Machine, Decision tree and k-Nearest neighbour algorithms were used, all of which were fine tuned to the problem. The first limitation was to obtain maximum possible accuracy and reliability of the phishing sites identification, the differences between the phishing and legitimate sites may be rather delicate and look like a genuine site. For models, the best model chosen was SVM with degree 9 polynomial kernel and C=2.5 for the regularization parameter getting the accuracy of 85.71/100%. This result shows that the SVM model is able to correctly classify phishing URLs, but then again, there is still significant room for improvement for the model in terms of accuracy.

Based on the analysis of using SVM, (Anupam, 2021) studied phishing website detection with nature-inspired optimization algorithms such as Bat Algorithm, Firefly Algorithm, Grey Wolf Optimizer, and Whale Optimization Algorithm. These methods seek to reduce error and improve the performance of SVM by selecting the right hyperplane to use in isolating phishing and legitimate websites using the URL characteristics, for example, length of IP address and availability of HTTPS. The Firefly Algorithm showed a performance between the best and the worst identified here as the Grey Wolf Optimizer significantly improved SVM performance to traditional Random Forest models optimized by Grid Search. This approach showed that optimization algorithms that imitated the natural phenomenon could be rather useful for fine-tuning the models used for phishing detection.

In another study, (Julio Lamas Piñeiro, 2022) developed an easy-to-use web-based phishing detection system that depends on machine learning models to classify URL as phishing or not, due the increase in phishing attacks during coronavirus outbreak. The concept involves creating and testing of three machine learning models including Random Forest, Classification Trees and Support Vector Machine with the processed web addresses aiming at determining the chance of phishing. For improved integration, an API was developed to ensure that a front-end HMI exists where users choose their desired model for the prediction. It also guarantees that anti-phishing tools are accessible by ordinary users of the Internet. One was model complexity – the problem of achieving high accuracy while keeping things simple so that even non-technical users could understand them. It was found that out of all the models, Classification Trees gave the highest results with an accuracy rate of 80% and therefore if we have to rely on one model for the end users, Classification Trees would be most accurate in this scenario.

Another study suggested analysis of machine learning techniques to enhance phishing website detection, focusing on improving classification accuracy with three key models: New methods like an improved Random Forest classifier, Support Vector Machine (SVM) and Neural Network with backpropagation; which are shown in (S. Sindhu, 2020). Both models were applied separately for assessing the ability of judging phishing websites among the visually similar URLs which is a practice being introduced with new technologies. The first difficulty solved was obtaining a high detection rate while keeping the ability of all the

compared algorithms equal. It was established that all the models gave outstanding results, with the SVM model giving the best performance, with 97.451% accuracy while the improved RandomForest was second at 97.369% joined by the NeuralNetwork at 97.259%. The results shown here suggest that phishing can be detected very effectively, especially with the help of machine learning and SVM in particular, although all the three methods demonstrate a high accuracy in detecting phishing websites.

2.3 Deep Learning Approaches for Phishing Detection

In recent years, deep learning approaches to identify phishing messages have relied even more on more sophisticated models to achieve high accuracy. The most significant approaches are labelled as MLP, LSTM, CNN, and certain patterns like BiLSTM and character-level CNN for the treatment of sequential and character-based features of phishing URLs. These models eliminate the need for enormous levels of human input and knowledge about the data set involved but can yield very high accuracy in a much shorter time, about 74.98% to 98.58% at times surpassing conventional machine learning methods. In conclusion, the performed research shows that deep learning is efficient in protecting against phishing threats.

More recently, (Aljofey, 2020) introduced a deep learning phishing detection model using character level CNN to scan and categorize URLs as phishing or legitimate without using the web content or any third-party service. Previous methods involve programming-based measures that rely on the identification of source code characteristics or third-party services that take more time to classify and need much pre-processing on features. To overcome these liabilities, the proposed CNN based model takes URL strings as input and learns characters sequentially, for the efficient identification of phishing without requiring any insight of the phishing techniques. Another problem was in attaining high levels of classification accuracy of phishing without having to design new features by hand. The model was then compared with the traditional machine learning models as well as the deep models using the hand-crafted, character embedding, character level TF-IDF and character level Count vectors. This study revealed that the proposed CNN model yielded high accuracy, 95.02% on the collected dataset, and 98.58% on benchmark datasets, MachineLearning 95.46%, and FuzzyML 95.22% Outperforming existing phishing detection models.

There is a study which have given by (Ullah, 2020) who developed a new model for filtering out spam messages by analyzing textual data of the body of the email. The proposed approach consists of Word Embeddings and a Bidirectional Long Short-Term Memory (Bi-LSTM) network to recognize the sentimental and sequential characteristics of the texts efficiently. As a result of CNN combined with Bi-LSTM network, for training efficiency as well as for higher-order text features extraction, the model. The problem addressed in this research relate to the flood of spam messages which end up flooding the mailboxes of the user hence the need for proper identification methods. The authors then test their model using the Lingspam dataset and the Spam Text Message Classification dataset to measure their model performance with regard to recall, precision, and F-score. Altogether, it can be stated that the results obtained prove the effectiveness of the proposed model at the degradation level of 98-99% and emphasizing that the new approach exceeds results of popular machine learning classifiers and state of art approaches in spam detection, while providing a reliable key to combating spam-related problems.

Another research given by (E. O. Asani, 2024) who aims to develop an application programming interface (API) for detecting phishing attacks by employing two advanced machine learning models: Among which, Multi-Layer Perceptron (MLP) and Long Short-

Term Memory (LSTM) were chosen. The MLP model has two hidden layers and also has adaptive learning rate whereas the LSTM is have designed especially for sequence data analysis. The problem discussed in this research is related to the fact that phishing becomes much more nuanced and, as such, requires better identification solutions. To this end, the authors gathered and cleaned a large, labelled data set and performed preprocessing on the data by eliminating punctuation marks, lap lower-case letters, English stop words stemmed and one hot encoding of tokenized words into binary vectors. Theses vectors were required to be padded in order to enhance the sequence length appeasing the model requirements. Accuracy in rating models' performance was assessed using performance evaluation measures including precision, recall and F1-score. As observed, for the MLP model the accuracy is as high as 98.8% and for the LSTM model, it is 96.8%. These outcomes confirm the efficiency of the developed server-side phishing detection API, and therefore pinpoint its potential as a significant tool for improving the protection of emails from cyber threats.

A new study that has done the study named Bidirectional Long Short-Term Memory based Gated Highway Attention Block Convolutional Neural Network (BiLSTM-GHA-CNN) is claimed in (Nanda, 2024) to improve the detection of phishing URLs, which look like the genuine websites of business and government organizations but are created to embezzle the users' personal information. The concern highlighted in this research is the inability of the presented anti phishing approaches to extract relevant features thus possibly misclassifying a given data and overly complicated algorithms which slows down their response. Concerning them, the proposed model uses BiLSTM to consider contextual aspects and CNN for salient aspects; at the same time, a highway improves convergence. In order to further improve the output features of CNN and BiLSTM, a gating mechanism is incorporated. The study reveals that with the proposed BiLSTM-GHA-CNN method, the accuracy of prediction is higher than other existing techniques coupled with better precision, recall and F1-score the overall response time is faster which is 12.46 ms only.

Finally, an email phishing detection framework provided by (R. Alotaibi, 2020) who has proposed CNNPD promising to use Convolutional Neural Networks (CNN) to analyze incoming emails as phishing or legitimate. This research seeks to fill this gap with an evaluation on the efficiency of phishing detection systems given the high impact and rate at which email phishing attacks that compromise data in both private and governmental organizations. The problem is the fact that there are different styles of phishing attacks, and the creation of the plausible-looking content is vital, while the creation of an efficient detection algorithm is quite hard to do due to the abovementioned reasons. Current techniques involve the use of having to manually extract the features, this usually involves help from a domain expert in deciding which features are of value hence time-consuming and expensive. CNNs used in CNNPD make it possible to automate feature extractions which in return improve detection. A trial of the framework on an email dataset fully shows benefits in the criteria like accuracy, precision, and recall, showing better results when compared to other similar methods and thus exhibits how it can be used to defend against the risks of email phishing.

2.4 Dockerization and Cloud Deployment

The Dockerized web application's performance was evaluated in research conducted by (Kurniawan, 2023), where it was deployed on various cloud platforms such as AWS, Microsoft Azure, and GCP. Open-source tools such as JMeter and SysBench were used to compare the deployments based on several metrics, including latency, throughput, CPU utilization, and database efficiency. The results showed that AWS performed better overall,

excelling in minimal latency and enhanced parallel processing. Azure outperformed in database and memory performance, while GCP demonstrated the best results in file I/O and load testing. This evaluation concludes that each cloud platform offers a different set of benefits, and the ideal cloud provider for Docker image deployment should be determined after carefully considering the specific performance requirements of the developers and users.

The emergence of containerization technologies such as Docker has been critical for deploying applications on cloud that are required to be scalable and efficient in their processes. Docker is better than conventional methods and techniques for the purpose of resource utilization such as hypervisors. This makes Docker the perfect technology to deploy applications in a microservice environment, as it reduces the costs incurred while deploying and conducting essential operations to meet the service requirements (Xili Wan, 2018). Docker platform is particularly useful in implementing Machine Learning Projects as the created Docker image is portable, which enables cross platform compatibility and seamless task management such as testing and training. The resource demands of Docker images might be higher than that of a regularly deployed application due to its complex file structures, mitigating this shortcoming requires finetuning and optimizing the cloud deployments of ML applications (Moses Openja, 2022).

3 Research Methodology

Based on the above literature research outcomes, a fool proof robust methodology has been derived around these studies to build a Phishing URL detection Flask Application to deploy its equivalent docker image on Aws and Azure for performance evaluation. The methodology of this research work was devised in the below phases.

3.1 Feature Extraction and Deep Learning

Phase 1 constitutes the Feature Extraction and Deep Learning Part, involving extracting the features from the data sourced from Kaggle for building the final dataset, preprocessing that data for removing irregularities including any outliers and finally training the models using deep learning algorithms to obtain accurate results.

3.1.1 Dataset collection and description

Beginning with the dataset collection, for this research the required dataset was obtained from Kaggle, (URL_Dataset_link) a collection of 651,191 URLs to help machine and deep learning models identify connecting to malicious sites. It includes four categories of URLs, benign, phishing, defacement, and malware attacks. The benign URL list contains 428103 entries, defacement URLs list 94457, phishing URLs list 94111 and malware URLs list 32520. The URLs were collected from multiple sources, including ISCX-URL-2016 which contained both benign and malicious URLs from Malware Domain Blacklist; Phishtank; PhishStorm sites. As this dataset caters to the requirement of this analysis, it is chosen in training of deep learning algorithms to detect phishing URLs and minimize interaction by users. This project will only utilize benign and phishing class URLs to be further feature extracted and used in the models.

3.1.2 Libraries Imported

For Deep learning various essential libraries were imported for Data processing, and visualization. The basic data management is supported by numpy and pandas while seaborn, matplotlib and plotly within its express and graph objects tools are responsible for data visualization. Sklearn, offers techniques for constructing models such as ensemble, RandomForestClassifier, ExtraTreesClassifier and summaries the outcomes with confusion matrix, accuracy score, and classification report. For data preprocessing, LabelBinarizer, LabelEncoder, MinMaxScaler are used. Tensorflow keras is used in deep learning models and their available layers includes conv1d, dropout, flatten, dense, bidirectional and lstm units. Other utilities include pickle for model save/load and train_test_split for splitting the data, allowing for complete model train & test.

3.1.3 Feature Extraction

Feature Extraction, an automation process of extracting the required characteristics and attributes directly from the raw data by learning specific sequences, patterns, and relationships is applied to obtain the final labelled fields for better accuracy and adaptability. Here the Features extracted from URL data includes three main categories: Address Bar, Domain, and HTML & JavaScript-based features. Address bar features include the length of the URL, the occurrence of "http/https" tokens, special characters like '@', and other specific patterns like hyphens, dots and slashes in the domain part of the URL. While they serve to determine the tendency towards phishing other features within the domain such as website traffic, age, and expiration date are used to determine the website's credibility. For Functionality based features, HTML & JavaScript tags are used to target features such as iframe redirecting, status bar modification, right click disabling, site forwarding, and internal, external, and Google links. Collectively, they provide the most holistic view of the prospects of detecting phishing indicators.

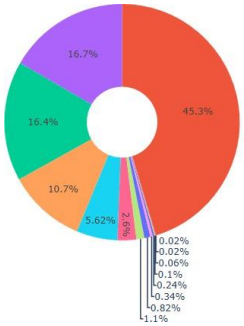
3.1.4 Data Preprocessing

Data Preprocessing is a methodology employed in Machine and Deep Learning models to clean the initial raw data in to more sustained usable form that is easier for the machine to understand. Any empty, negative or null values along with outliners are removed before proceeding to train the models. In this experiment, during data preprocessing, the categorical data are converted to numerical form to be used with the deep learning algorithms for improved compatibility and model interpretability. The dataset variables are divided into feature variables (X) and the output variable (Y), where X excludes the label and Y is the "Label" column. Target variables in categorical form are encoded using the LabelEncoder and the to_categorical function, numerical features are scaled using MinMaxScaler or StandardScaler to ensure features are on the same scale. As a result of preprocessing, the final dataset includes 5,000 rows and 18 features ready for training the model, with more balanced scaling and categorical feature transformation, contributing to the better performance of the model.

3.1.5 Data Visualization

Data visualizing methods are essential in the comprehension of the relationships between the URLs present in the dataset, two of those techniques used in this research are, a URL depth pie chart and a correlation matrix. The pie chart in the Figure 1 illustrates the

segregation of the URLs based on the URL depth column in the dataset. The largest section of the pie chart corresponds to the value “1”, where 43.3% of URLs lie in this section, 16.7% of URLs lie under the value “3” and so on. Figure 2 displays the correlation matrix which highlights the relationships between the features present in the dataset. The features that present in the dataset that is directly proportional with any other field for its presence has a positive value and features that has no correlation with each other has negative values beside it. The features “Have_At” and “Have_IP” have positive corelation and the features “Redirection” and “IframeRedirection” have a negative corelation, this greatly assists in further feature analysis and in improving the accuracy of detecting the Phishing URLs.



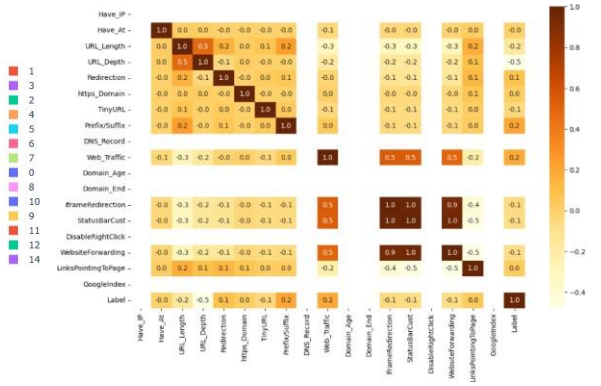


Figure 1 and 2: The URL depth Pie chart and the Corelation Matrix

3.1.6 Data Splitting for Model Training

The dataset in question is split into train test and validation sets with a 80:10:10 ratio in order to have fair division of data for model training. First, the function train_test_split is employed to split the data into a training and test data, where test_size was set to 0.1 and stratify parameter as Y. The entire training data is further divided into training and validation by using train_test_split separately with test_size = 0:10 and stratify = y_train, which gives 80:10:10 of training, validation and testing data. This split benefits the model by helping the tuning process and allowing for a logical separation of the data for unbiased testing.

For the model training, after thorough understanding and evaluation of the strengths and weakness of various Deep Learning algorithms, CNN, LSTM and BiLSTM were choose for this study for handling the complex data patterns and phishing detection. The Strength of Convolutional Neural Networks (CNNs) models are that they are well suited to easily capture the complex patterns and dependencies of the URL string structure and to deal with the characteristics of sequential data, Long Short-Term Memory (LSTM) a recurrent neural network model is used for accurate prediction. Finally, the Multi layered Bidirectional LSTM (BiLSTM) with two LSTM Layers are used to process the input URL string formats in both forward and backward directions for accuracy. Together all these models address the key challenges and lay a strong foundation in building a robust Phishing URL detection system and the model BiLSTM with the best accuracy is chosen for flask application.

The Deep Learning code for the flow discussed above is well structured and documented in the Colab link : DL_Model_Training_Colab_Code.

3.2 Developing and Dockerizing the flask Application

Phase 2 deals with the creation and dockerization of the Flask application, in this phase the methodology adopted uses a lightweight framework called Flask. Flask is a Web Server

Gateway Interface Python framework that is easy to use without any dependencies and can scale to balance the load parallelly. This flask app is then containerized using docker, a platform that facilitates the packaging of code along with its dependencies for ease of probability across multiple different environments.

Within the project directory, for the python version 3.10.11 installed, the Framework Flask was installed from the package manager using pip install Flask command and imported in the python application main file "app.py", the routes to display the pages of the website were also configured in the same file. Other front-end libraries such as Bootstrap, font awesome and JavaScript plugins were set up for designing the user interface of the Flask application. The website contains two pages, Home Page and Service Page. The Home Page captures the introduction and models used in the experiment and the Service Page allows the users to input the URLs to detect phishing with a click to action button.

Figure 3 and 4: Flask Application UI

For the above flask Application docker images are created and deployed on respective clouds where the image is built by maintaining a python 3.9 slim base image along with packaging the contents and dependencies required for build. AWS facilitates docker image creation without any manual configuration, once the Dockerfile and application code directory is maintained and pushed on to the GitHub Repository. Choosing the docker platform on the Elastic beanstalk environment will automatically detect and builds the flask application in a docker container. On the other hand, in Azure the image is build using docker desktop, to create the docker image "docker build -t purldwebapp0" command is used and then authenticated with Azure CLI by entering the login credentials, once the login is successful, the image is pushed to the Azure Container Registry (ACR) by creating a tag for ACR. Finally, the image stored in the ACR is then deployed to various Azure Container Apps.

GitHub Repository Link: https://github.com/Yamini-Murugan/URL_Phishing_Detection_App

4 Design Specification

For the research in question, the design specification section pins down the architecture discussing the workflow and tools used in this research.

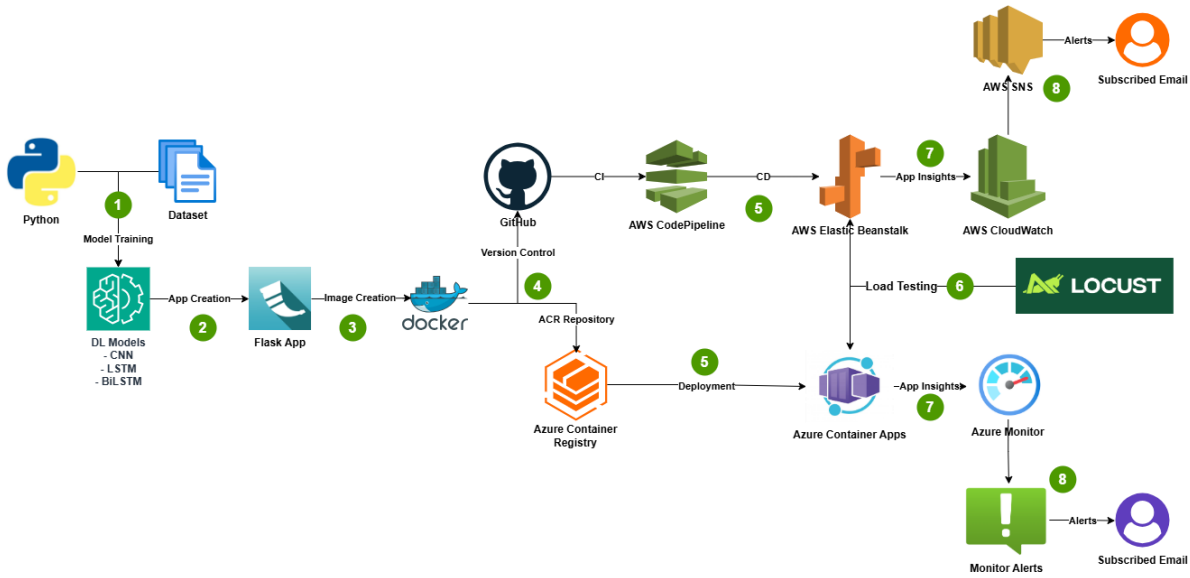


Figure 5: Architecture Diagram of proposed system

The Architecture flow of the phishing URL detection system begins with the model training (1), here the process starts with the raw URL data being collected and feature extracted, necessary for preparing the datasets required for training the phishing detection models. The data is then subjected to filtering and cleaning to handle and remove any irregularities. While the data preprocessing step handles the missing values, feature extraction transforms any categorical data into numerical formats to enhance the use of datasets. Exploratory Data Analysis (EDA) helps in visualizing the features to identify the key characterizes with plot graphs and understand the relationships between each field with co-relation matrix. Next, the classification models comprising of deep learning algorithms such as CNN, LSTM, and BiLSTM are introduced to classify the URLs as benign or phishing with their ability to accurately recognize complex patterns and sequences in the URLs. Following the model training, the results are evaluated through confusion matrices and classification report to determine the best performing models. In Step (2) the flask Application is created using Flask framework with an API endpoint and underlying BiLSTM Model to classify the URLs in real time. Followed by step (3), docker image creation, a crucial step for containerized deployment. The Docker image after initializing the base image captures all the dependencies and libraries from a “requirement.txt” file for facilitating the deployment on Azure and AWS platforms. In Step (4) the required code setup containing the docker file is pushed to GitHub for triggering the CI/CD Pipeline on AWS, Subsequently for Azure, the docker image is maintained in Azure Container registry. In Step (5) the deployment for AWS is pipelined through AWS CodePipeline is set up, AWS Elastic Beanstalk is set as deployment platform. On the other hand, in Microsoft Azure, the Azure Container Apps handles the end-to-end deployment of the containerized Docker image of the Phishing URL detection Flask application effortlessly with the help of other Azure tools in the form of Azure Container registries as the source of the Docker image. In the testing stage (6) Locust is used to generate the required load on both the platforms to measure performance, in the background the metrics are collected in step (7) using AWS CloudWatch and Azure Monitor to record the key metrics concerning the application health and performance. These metrics and logs collected by these native tools are extremely valuable insights to keep the monitoring in place and trigger any alarms in case of any bottlenecks. This is captured in the final step (8) where CPU utilization Alarms are configured in AWS and Azure to trigger timely notification through Amazon Simple Notification Service and Azure Metric Alters.

5 Implementation

The implementation comprises of the deployment strategies (phase 3) employed on both cloud platforms for the Dockerized application. On Aws, the process of continuous integration and continuous delivery of the code build, test and deploy is achieved through CodePipeline and with Elastic Beanstalk, the entire flow is simplified for better management of the application. On Azure, ACR Azure Container Registry is used to store and manage the docker image while the Azure App Service facilitates the seamless deployment of containerized application with autoscaling and metric monitoring features.

5.1 Cloud Deployment on AWS Using Docker

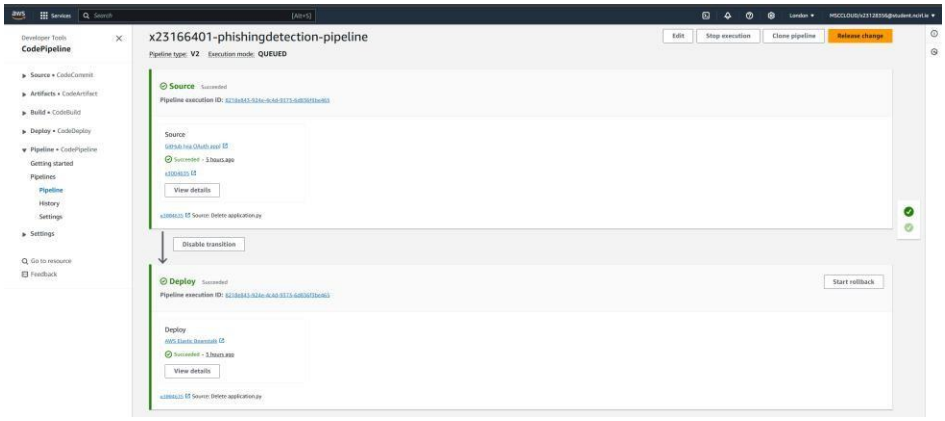


Figure 6: AWS CodePipeline Setup

Another part of the Cloud implementation of the application is the AWS deployment. Here, a CI/CD pipeline is set up using the AWS CodePipeline service. The GitHub repository containing the application source code and the Dockerfile is provisioned to be the source for the pipeline as it simple to administer version control this way. The Code pipeline is triggered whenever the code is updated and pushed to GitHub. This process ensures the whole pipeline is automated and the code integration is streamlined. Human intervention is totally eliminated which ensures no drop in quality of the process and only the successful build stages are proceeded to the next stage.

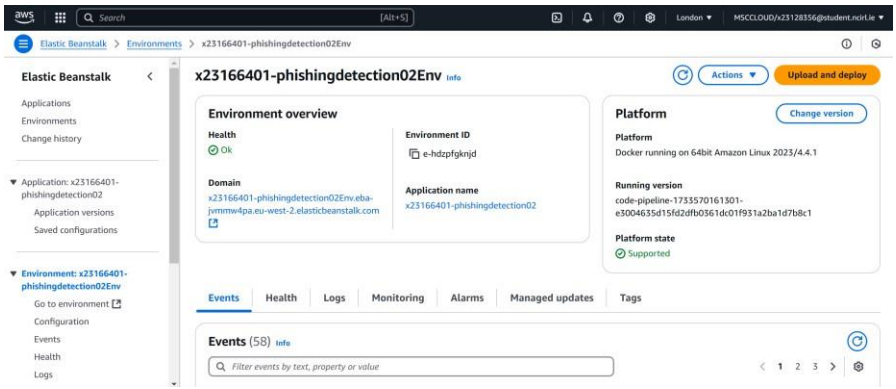


Figure 7: AWS Elastic BeanStalk Environment

The second phase of the AWS deployment in the Continuous Deployment implemented using AWS Elastic Beanstalk. The Elastic Beanstalk environment was chosen run on the Docker platform running on Linux to be compatible with the application configurations. The Docker image is automatically built by the underlying EC2 instances with is provisioned simultaneously when the application is deployed. This is done using the Dockerfile in the GitHub repository. The EC2 instances are managed by the deployment service and this provides the much-needed fault tolerant and load balanced environment for the application. The deployment process is entirely automated and the whole Ci/CD pipeline enhances the development efficiency with minimal manual oversight.

AWS Deployed Link: http://x23166401-phishingdetection02env.eba-jvmmw4pa.eu-west-2.elasticbeanstalk.com/

The monitoring service of AWS CloudWatch is set up for collecting the essential metrics of the application while it is under load, where the CPU usage, Latency, Network I/O and load Average metrics are collected. To ensure that the CPU threshold is not exceeded by the application, a alert system is set up using AWS SNS, where a alert is configured to be issued to the subscribed Email ID if the CPU utilization percentage goes above 60% over a time period of 5 minutes. The configuration of this alert can be seen in figure (8) below.

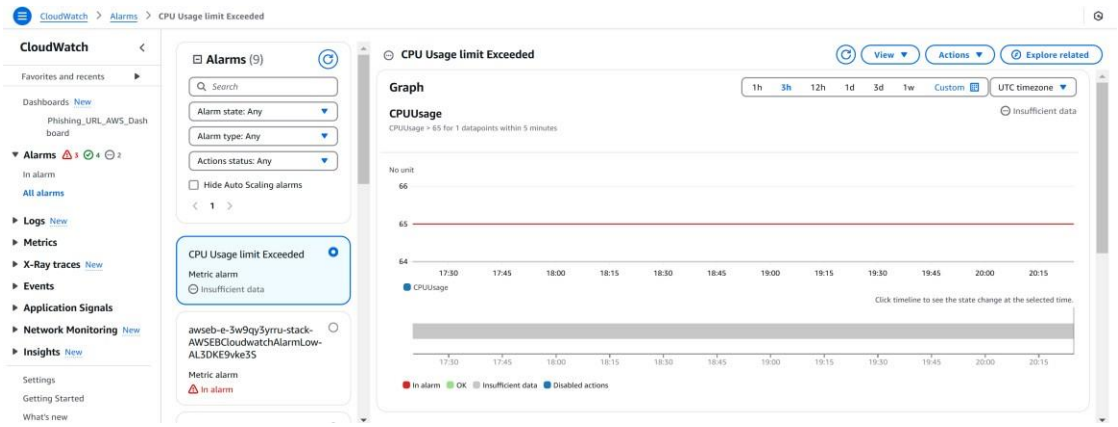


Figure 8: AWS SNS Alerts Set up

5.2 Cloud Deployment on Azure Using Docker

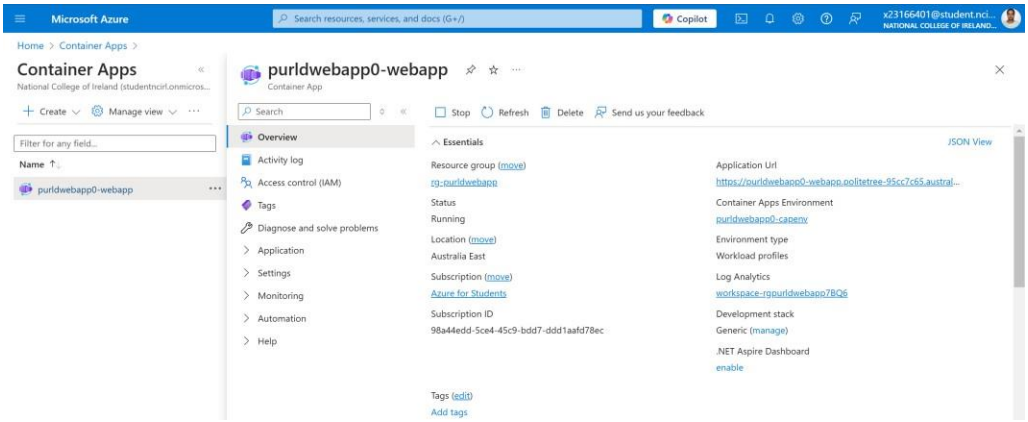


Figure 9: Azure Container Registry containing the Docker image

The cloud deployment phase of the Phishing URL detection Flask application for Microsoft Azure was done implementing the following methodology. The Docker Image that was created on the local device was pushed to the Azure Container Registry using the Azure CLI. The application was containerized into a Dockerfile, with the requirements of the environment specified in detail along with its dependencies and the instructions to deploy the application on the runtime environment. In the next stage, the Dockerfile was tagged by following the naming conventions of ACR to easily reference the image in the further stages. Azure Container Registry was used for this process as it allows for stronger integration with other Azure services and enables seamless automation in the deployment process during version control. The Docker image which was stored in Azure Container Registry is then deployed in this stage on Azure Container Apps. The Source for the deployment process was set as Azure Container Registry during setting up the configurations for the Container Apps environment. The tag that was assigned to the Docker image was referenced to get hold of the container's image and the version. Various parameters were assigned such as the resource allocation settings like CPU and memory limits, scaling parameters and the environment variables were defined. The Container Apps then automatically pulls the image from the Container Registry's repository, deploys in a container and provides an endpoint for the users to access it. This whole process is fully automated and the serverless environment possesses built-in monitoring services for container surveillance purposes. The Continuous Delivery support is enabled due to its integration with other Azure services, making it the ideal platform to deploy an application of this magnitude.

AZURE Deployed link: https://purldwebapp0-webapp.politetree-95cc7c65.australiaeast.azurecontainerapps.io/

In a comparable manner, an alert system is configured on Microsoft Azure using Azure Monitor to track application performance metrics under load. Essential metrics such as CPU utilization, latency, network traffic, and load average are continuously monitored. An alert rule is established to trigger if CPU utilization exceeds 60% over a five-minute period. When this threshold is reached, a notification is sent via Azure Notification Hub to the designated email address, enabling timely awareness and prompt response. This parallel configuration across both cloud platforms ensures comprehensive monitoring and proactive issue management, thereby enhancing application reliability and performance.

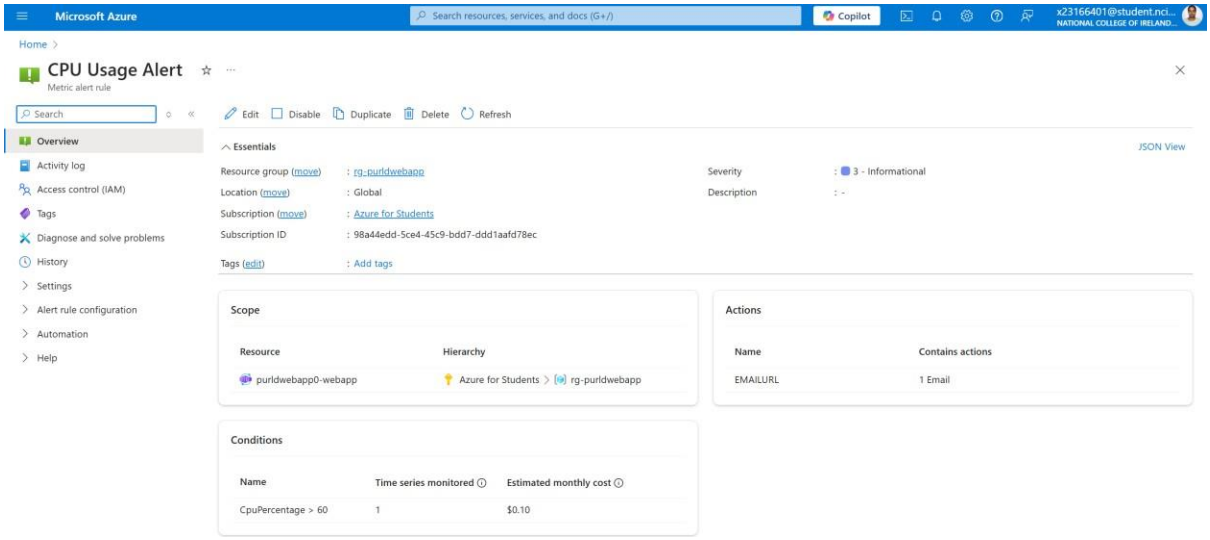


Figure 10: Azure Monitor Metrics Alert setup

In a comparable manner to AWS, Azure Monitor alerts has been used configure an alert system for the scenario where the CPU utilization crosses 60% which can be seen in figure (10) above, when this limit is reached or breached, an email to the subscribed user is sent using the Azure Notification Hub. This provides an opportunity to monitor the resources allocated to the application closely and aids greatly in cost efficiency.

6 Results and Evaluation

Case Study 1: Evaluation of Machine Learning Models

Three different Deep Learning models were considered while developing a Phishing URL detection application, namely, CNN, LSTM, BiLSTM. The accuracies achieved by these models during model training were 80%, 84%, 85 respectively and The BiLSTM model was chosen to create a Flask application to detect phishing URLs. This model's superior performance may be attributed to its capability to record the whole input sequence of the URLs to efficiently detect the URL patterns. This is due to its bidirectional architecture which has enabled it to comprehend the input data from both directions which enhances the ability to accurately detect non-linear relationships of different URL sequences. These factors make BiLSTM the best model for data analysis.

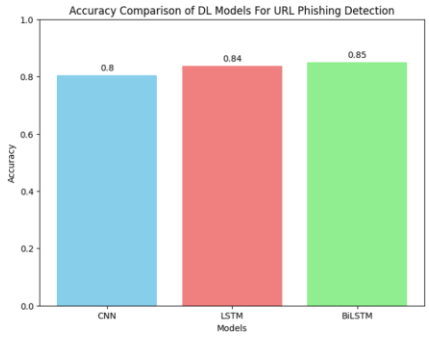


Figure 11: DL Models accuracy scores

Case Study 2: Experimentation of Load Testing on AWS and Azure

The deployed application on AWS and Azure was load tested using Locust with the parameters set at 2000 users accessing the site at the rate of 5 users joining per second, which yielded the following results: AWS had slightly lesser response times at 60000 milliseconds for users up until 50th percentile and 95000 milliseconds for users up until the 95th percentile. The response times values for Azure were 90000 milliseconds and 150000 milliseconds for the respective percentile values. Another key insight observed was, AWS's response were not consistent as the latency fluctuated often leading to an unstable release of the application by Elastic Beanstalk, which was not the case with Azure's Container Apps, where the latency was even throughout the process. This indicates that Azure can handle higher loads without anu interruptions and the users will be able to experiences a smooth connection throughout their interaction with the application unlike AWS. The service reliability during peak traffic is better handled by Azure and it is suitable for applications that expect high incoming users.

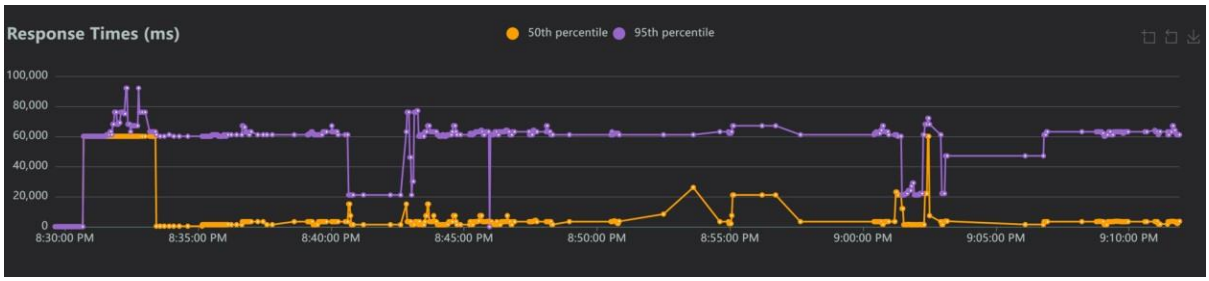


Figure 12: AWS Locust Response times

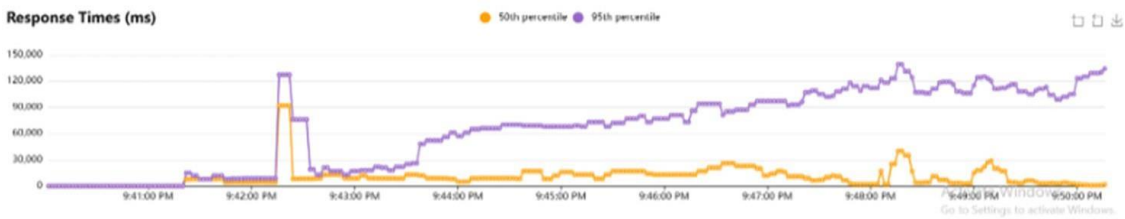


Figure 13: Azure Locust Response times

Comparison of Network In (Bytes) Between AWS and Azure Infrastructures

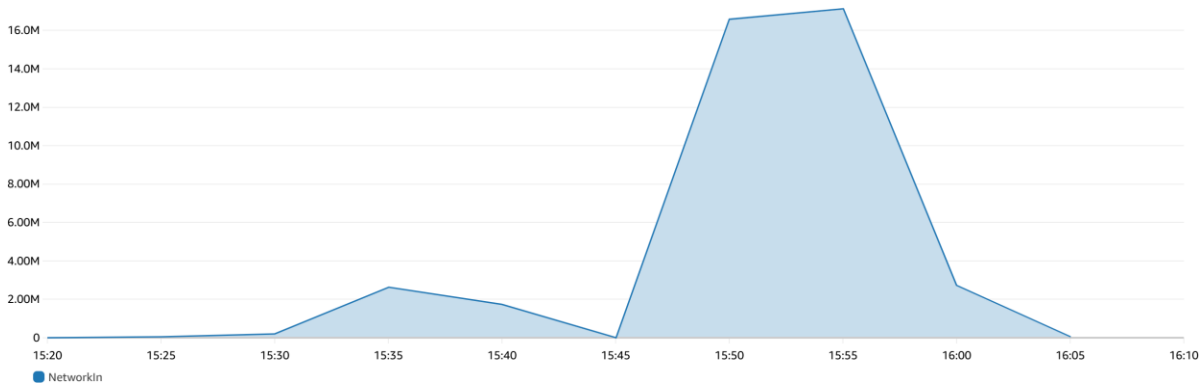


Figure 14: AWS Network In Graph

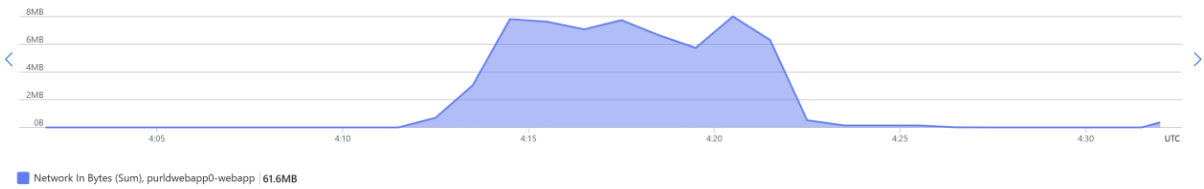


Figure 15: Azure Network In Graph

The load testing of the cloud environments done using Locust revealed the Network In values of both the platforms, which is 61 million Bytes on Azure and 16 million Bytes on AWS. A higher network In value on Azure indicates that Azure possesses grater capacities to handle incoming traffic, pointing towards greater scalability and workload balance. Microsoft Azure's robust infrastructure and data handling efficiency is highlighted by this large difference b/w the readings on both platforms. This makes Azure more suitable when it comes to applications that experience high-traffic and load.

Comparison of Network (Out Bytes) Between AWS and Azure Infrastructures

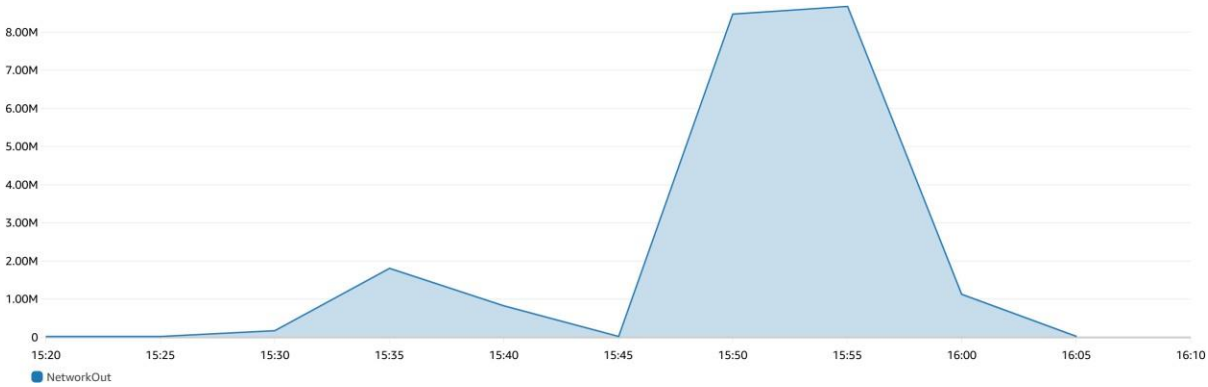


Figure 16: AWS Network Out Graph

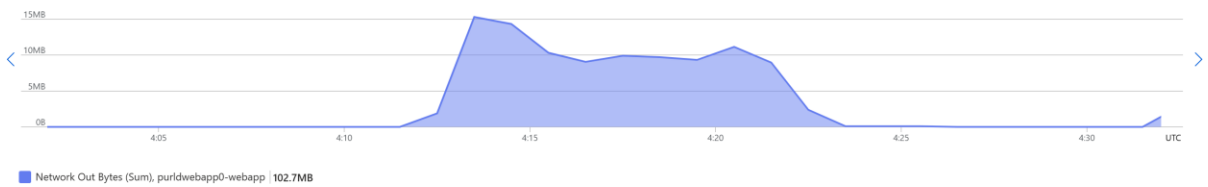


Figure 17: Azure Network Out Graph

The load testing of the outbound data in the form of Network Out revealed that Azure processed 102 million Bytes and AWS, 8 million Bytes. This suggests that Azure has higher data processing capabilities. This can be attributed to Azure's superior data transfer abilities that is finetuned to deliver large amounts of data to its end-users and clients. Data-intensive applications that incur high data transfer rates should be implemented on Microsoft Azure as it has proved more reliable for these tasks in this test.

Comparison of Latency Between AWS and Azure Infrastructures

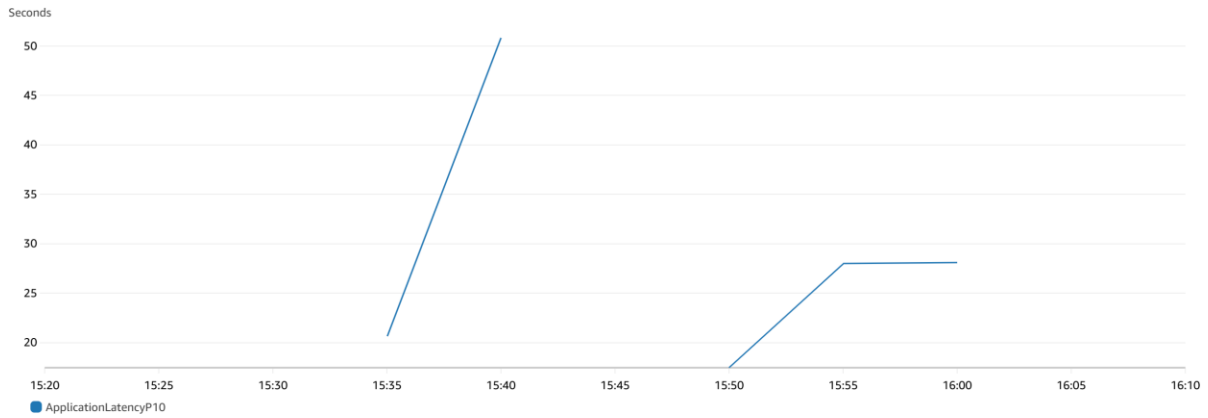


Figure 18: AWS Latency Graph

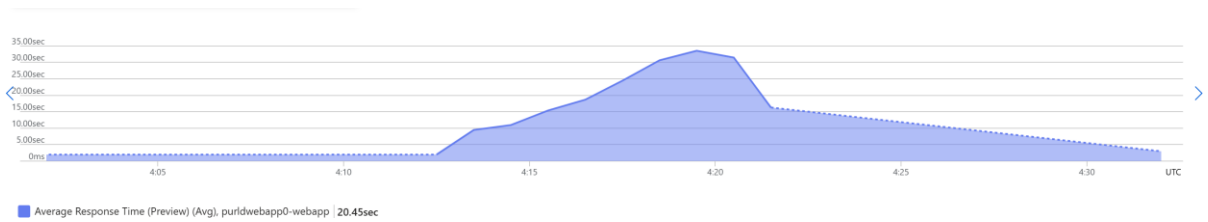


Figure 19: Azure Latency Graph

The Latency that was recorded on both platforms were 35 and 50 seconds on Azure and AWS respectively. The average latency was around 20 seconds on Azure and 25 Seconds on AWS, another observation is that the latency values were similar throughout the whole load testing process on Azure, but it was not the case on AWS. The consistency displayed by Azure has assured that it has better abilities to handle and serve several user requests in a timely, reliable manner. Azure can be preferred for the deployment of applications that require predictable, timely and stable response times.

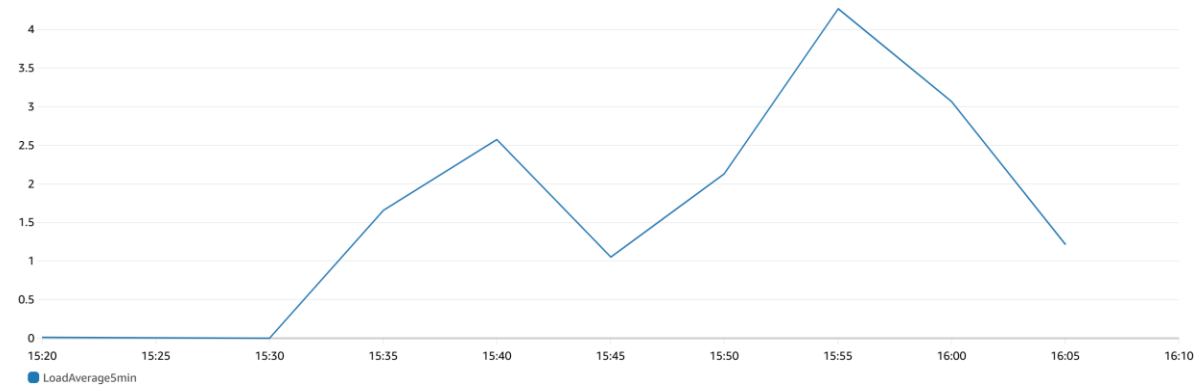


Figure 20: AWS Load Average Graph

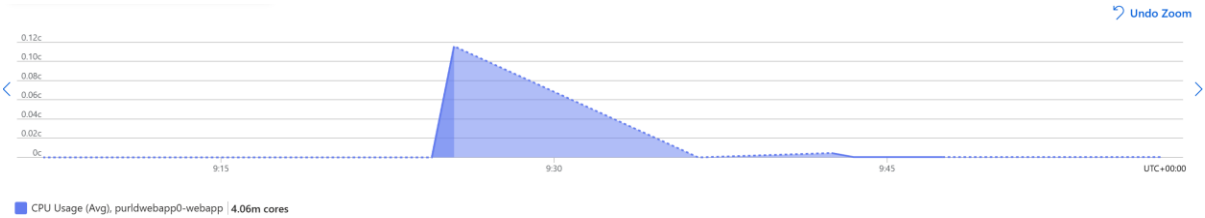


Figure 21: Azure Load Average Graph

Comparison of CPU Utilization Between AWS and Azure Infrastructures

The test conducted to test the CPU utilization values in terms of no. of cores utilized has shown that the AWS used 4 cores at its peak and 2.5 cores on an average to process the incoming the data, the values that Azure displayed were the utilization of 4 cores and 3 cores at its peak and on an average respectively. This indicates that Azure exhibits more efficient usage of the CPU cores available by smartly distributing the incoming load between a greater number of cores, suggesting streamlines processing ability and better workload management. Underutilization of the processing power can be seen in the case of AWS's results. Platforms that require more computational power and efficient power handling may be more suited and may possess better performance when deployed on Microsoft Azure rather than on AWS.

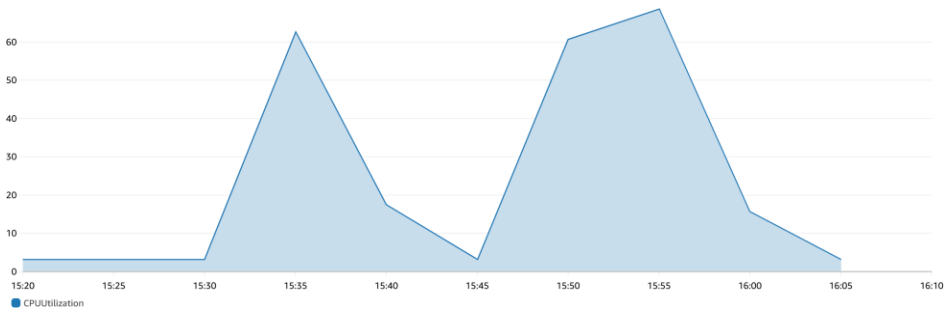


Figure 22: AWS CPU Usage Graph

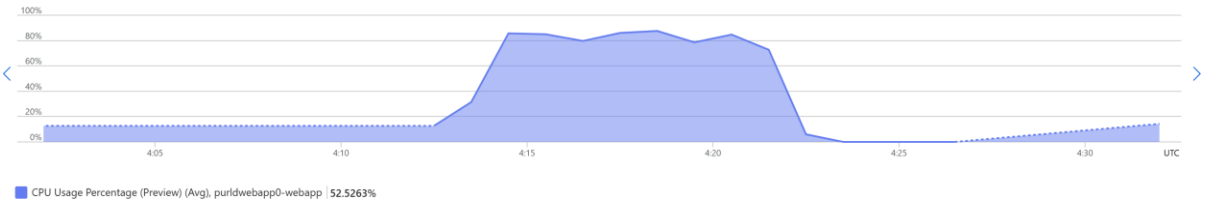


Figure 23: Azure CPU Usage Graph

The CPU Utilization of Azure to process the incoming requests was 84% at its peak and was stable at 52% on average, while it was 65% and 40% for AWS on peak and average values. This means that Azure was able to handle intensive workloads more effectively by smartly assigning its processing resources to handle the load. On the other hand, complete utilization of its CPU capacity was not achieved by AWS, this may lead to underutilizing its power and delivering subpar performance when under stress. Azure can be preferred to deploy resource heavy applications.

Metric	AWS	Azure
Hosting Environment	AWS Elastic Beanstalk	Azure Container Apps
Users Simulated	2000 users @ 5 users/second	2000 users @ 5 users/second
Response Times (50th & 95th Percentile)	Vary between 40 ms and 80 ms	Vary between 30 ms and 120 ms
Performance Observations	Moderate response times with some spikes; steady load handling	Higher variability in response times; handled higher RPS fluctuations
Network In (Bytes)	16 Million Bytes	61 Million Bytes
Network Out (Bytes)	8 Million Bytes	102 Million Bytes
Latency	Average - 20 ms, Max - 25 ms	Average - 35 ms, Max - 50 ms
Load Average (Cores)	Average - 2.5, Max - 4	Average - 3, Max - 4
CPU Utilization (%)	Peak – 65%, Average – 40%	Peak – 84%, Average – 52%

Table 2: Comparison of Locust Load Test Results on AWS and Azure

7 Conclusion and Future Works

In this research project, a Flask application for Phishing URL detection based on BiLSTM model was successfully created. This was then Dockerized and deployed on AWS and Microsoft Azure utilizing different deployment pathways. The docker image was stored in a repository on Azure Container Registry and was deployed on Azure Container Apps, while the application codebase was pushed to GitHub to be deployed through a CI/CD pipeline on AWS. To do this, AWS CodePipeline is used here for Continuous Integration and AWS Elastic Beanstalk for Continuous Deployment. For the purpose of monitoring of performance metrics, Azure Monitor and AWS CloudWatch has been integrated with the respective deployments. Load testing is undergone by the applications with the help of Locust to collect performance metrics such as Response times, CPU usage, and metrics related to Latency and Scalability. This comprehensive analysis was done for determining the most suitable cloud service provider for deploying a Dockerized Application. The performance comparison revealed that Azure is better at handling large amounts of data and has high network throughput whereas AWS has excelled at possessing faster response times, albeit a

bit inconsistent in that regard. Both the cloud platforms have streamlined, and robust deployment environments set up in accordance for enterprise-level setup and deployment. This study has considered various metrics such as latency, scalability and throughput of 2 different cloud service providers to determine the best platform for deployment. The metrics collected as a result of this research work are of paramount importance as they serve as the deciding factor in improving performance and preventing any bottleneck situations. These alerts configured on both the platforms trigger notifications via AWS SNS and Azure Metric Alerts allowing businesses to make timely actions and resolve any potential issues before they arrive.

Limitations and Future Work: This research could be further improved in several areas, starting from finetuning the Deep Learning models to be more accurate by using a larger and more diverse dataset and potential improvements to the model based on certain strategies and more marked future trends in new architectures. Another area of improvement is the utilization of ensemble learning algorithms, including stacking or boosting, which aggregate results of several models in order to prevent low accuracy. Further, the experiments can be performed with even more sophisticated architectures such as Transformer models or some CNN-LSTM hybrids to get the biggest boost. Multi-region deployment is another avenue which can be explored upon in terms of reducing the latency for regional users. Other orchestration tools like Kubernetes and services like Azure Kubernetes Service (AKS) or Amazon Elastic Kubernetes Service (EKS) can be used for scaling the application containerized systems for achieving better load balancing and improved management of resources. For applications with heavy backend processes with database, the choice of parameters for the purpose of performance evaluation can be broadened by including metrics such as throughput, error rates, and database performance which would provide valuable insights. The above suggested recommendations would further enhance the scope of this research in building a more advanced Deep Learning Dockerized applications on multi-cloud environments.

8 References

Adam Kavon Ghazi-Tehrani, H. N. P., 2022. *The New Technology of Financial Crime.* [Online]
Available at: https://www.taylorfrancis.com/chapters/edit/10.4324/9781003258100-3/phishing-evolves-analyzing-enduring-cybercrime-adam-kavon-ghazi-tehrani-henry-pontell

Alabdan, R., 2020. *Phishing Attacks Survey: Types, Vectors, and Technical Approaches. Future Internet 12.* [Online]
Available at: https://www.mdpi.com/1999-5903/12/10/168

Aljofey, A. J. Q. Q. Q. H. M. &. N. J., 2020. *An Effective Phishing Detection Model Based on Character Level Convolutional Neural Network from URL.* [Online]
Available at: https://www.mdpi.com/2079-9292/9/9/1514

Alkhalil Z, H. C. N. L. K. I., 2021. *Phishing Attacks: A Recent Comprehensive Study and a New Anatomy.* [Online]
Available at: https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2021.563060/full?ref=based.inc

Anupam, S. K. A., 2021. *Phishing website detection using support vector machines and nature-inspired optimization algorithms.* [Online]
Available at: https://link.springer.com/article/10.1007/s11235-020-00739-w#citeas

Carlo Marcelo Revoredo da Silva, E. L. F. V. C. G., 2020. *Heuristic-based strategy for Phishing prediction: A survey of URL-based approach.* [Online]
Available at: https://www.sciencedirect.com/science/article/pii/S0167404819301622
Diki Wahyudi, M. N. A. A. P. A., 2022. WEBSITE PHISING DETECTION APPLICATION USING SUPPORT VECTOR MACHINE (SVM). *JOURNAL OF INFORMATION TECHNOLOGY AND ITS UTILIZATION*, 5(2).
E. O. Asani, O. S. B. A. E. A. A. S. B. O. V. A. a. J. O. O., 2024. *A Server-side Phishing detection API using Long-Short Term Memory (LSTM) and Multi-layer Perceptron (MLP).* [Online]
Available at: https://ieeexplore.ieee.org/abstract/document/10629792
I. Kim, W. S. J. B. J. K. a. Y. W. C., 2023. *PCSF: Privacy-Preserving Content-Based Spam Filter.* [Online]
Available at: https://ieeexplore.ieee.org/abstract/document/10064347
Julio Lamas Piñeiro, D. L. R. W. P., 2021. *Web architecture for URL-based phishing detection based on Random Forest, Classification Trees, and Support Vector Machine.* [Online]
Available at: https://www.journal.iberamia.org/index.php/intartif/article/view/750
Kurniawan, A. P., 2023. *Performance Evaluation for Deploying Dockerized Web Application on AWS, GCP, and Azure.* [Online]
Available at: https://ieeexplore.ieee.org/abstract/document/10140775
Mohammad, R. M. T. F. M. L., 2013. *Intelligent rule-based phishing websites classification.* [Online]
Available at:
https://ietresearch.onlinelibrary.wiley.com/action/showCitFormats?doi=10.1049%2Fiet-ifs.2013.0202
Moses Openja, F. M. F. K. B. C. a. H. L., 2022. *Studying the Practices of Deploying Machine Learning Projects on Docker.* [Online]
Available at: https://dl.acm.org/doi/abs/10.1145/3530019.3530039
N. Q. Do, A. S. O. K. E. H.-V. H. F., 2022. *Deep Learning for Phishing Detection: Taxonomy, Current Challenges and Future Directions.* [Online]
Available at: https://ieeexplore.ieee.org/abstract/document/9716113
Nanda, M. G. S., 2024. *URL based phishing attack detection using BiLSTM-gated highway attention block convolutional neural network.*. [Online]
Available at: https://link.springer.com/article/10.1007/s11042-023-17993-0#citeas
R. Alotaibi, I. A.-T. a. F. A., 2020. *Mitigating Email Phishing Attacks using Convolutional Neural Networks.* [Online]
Available at: https://ieeexplore.ieee.org/abstract/document/9096821
S. Sindhu, S. P. P. A. S. F. R. a. M. S. A. N., 2020. *Phishing Detection using Random Forest, SVM and Neural Network with Backpropagation.* [Online]
Available at: https://ieeexplore.ieee.org/abstract/document/9277256
Simon Bell, P. K., 2020. *An Analysis of Phishing Blacklists: Google Safe Browsing, OpenPhish, and PhishTank.* [Online]
Available at: https://dl.acm.org/doi/abs/10.1145/3373017.3373020
Ullah, S. E. R. a. S., 2020. *Email Spam Detection using Bidirectional Long Short Term Memory with Convolutional Neural Network.* [Online]
Available at: https://ieeexplore.ieee.org/abstract/document/9230769
Xili Wan, X. G. T. W. G. B. B.-Y. C., 2018. *Application deployment using Microservice and Docker containers: Framework and optimization.* [Online]
Available at: https://www.sciencedirect.com/science/article/pii/S1084804518302273