Phisher.AI

Muhammad Muwahid Asim – 2019352

Advisor: **Dr. Farhan Khan** ***Co-Advisor:*** **Mr. Talha Laique**

Faculty of Computer Science and Engineering
Ghulam Ishaq Khan Institute of Engineering Sciences and Technology

Certificate of Approval

It is certified that the work presented in this report was performed by **Muwahid Asim** under the supervision of **Dr. Farhan Khan.**
The work is adequate and lies within the scope of the BS degree in Computer Science/Computer Engineering at Ghulam Ishaq Khan Institute of Engineering Sciences and Technology.

---------------------	---------------------
Dr. Farhan Khan	**Mr. Talha Laique**
(Advisor)	(Co-Advisor)

Dr. Ahmar Rashid
(Dean)

ABSTRACT

This research project explores the utilisation of deep learning and artificial intelligence (AI) models for cyberattacks. The proposed approach makes use of a character-wise one-hot encoded vector input and a convolutional neural network (CNN) and recurrent neural network (RNN) model. The results of the experiments demonstrate the effectiveness of the CNN-RNN model in mitigating the risks associated with these types of cyberattacks. Accuracy and the area under the receiver operating characteristic curve (AUC) are used in the study to evaluate the efficacy of the suggested strategy.

Test outcomes demonstrate that the suggested CNN-RNN model for phishing email detection obtained an accuracy of 87.59% and an AUC score of 88%. Another application that was explored is SMS spam detection and obtained an AUC score of 99.62% and accuracy of 98.54%. These findings show that the suggested strategy can be quite successful in reducing the risks involved.

Overall, this study emphasises how important deep learning and AI models may be in detecting and preventing intrusions. Utilising the strength of these models, people and organizations can take preventative action to guard against monetary losses, privacy invasions, and identity theft. Future studies could examine the efficacy of various deep learning models and assess the proposed strategy against various cyberattacks.

ACKNOWLEDGEMENTS

I would like to begin by expressing my heartfelt gratitude to Allah, the Almighty, for giving me the strength and guidance to complete this project. I am humbled by His blessings and grateful for His constant support throughout this journey.

I would also like to express my sincere appreciation to my advisor, Professor Dr. Farhan Khan for his invaluable guidance and support throughout this project. His insights and expertise have been instrumental in shaping the direction of the research.

In addition, I would like to extend my gratitude to my co-advisor, Professor Mr. Talha Laique, for his valuable contributions and feedback throughout this project. His guidance and encouragement have been invaluable in helping navigate the challenges and complexities of this research.

Finally, I would like to thank all those who have supported throughout this project, including my colleagues and friends. Your encouragement and support have been instrumental in helping me achieve my goals.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter I - Introduction

Overall, the research emphasizes how crucial deep learning and AI models may be in identifying and preventing intrusions. Utilizing the strength of these models, people and organizations can take preventative measures to guard against monetary losses, privacy invasions, and identity theft. Future studies could examine the efficacy of various deep learning models and assess the suggested approach against various cyberattacks.

Attacks involving email phishing are fraudulent efforts by cybercriminals to steal private information, including login credentials or financial data, by pretending to be a trustworthy company through an email. These attacks are becoming one of the most prevalent types of cyberattacks and are growing constantly.

Phishing attacks aim to deceive victims into downloading files, clicking on links, or entering their login credentials into a fake login page. Significant repercussions from these attacks could include identity theft, financial losses, and compromised computer systems.

To stop their spread and lessen their effects, it is crucial to identify these attacks as soon as possible. A critical component of phishing attack detection and prevention is artificial intelligence (AI). Machine learning algorithms are used by applications powered by AI to analyze email traffic and identify suspicious patterns and behaviors that indicate a phishing attack.

AI-based systems can swiftly and effectively identify phishing assaults and stop them from doing any damage by constantly adapting and learning to new threats. To alert security professionals and end users about potential dangers, these systems may additionally generate alerts and notifications. Attacks using email phishing present a severe risk to both people and businesses. Advanced security solutions that make use of AI-based detection and prevention strategies are necessary due to the rising frequency and sophistication of these attacks. Phishing attacks can be detected and avoided, helping to safeguard confidential data, avert financial losses, and preserve the integrity of IT systems.

In broad terms, adopting artificial intelligence (AI) techniques for phishing email is an essential step towards protecting private and company data in the current digital landscape. Individuals and companies can take preventative steps to safeguard against monetary losses, violations of privacy, and identity theft by utilizing AI to detect and stop such kinds of attacks.

Chapter II – Literature Survey

URL based Detection:

URL-based phishing detection is the process of identifying and preventing phishing attacks by analysing the URLs used by cybercriminals to trick their targets. Cybercriminals seek out sensitive information from innocent victims via phishing attacks, a type of social engineering attack that works by creating websites or emails that seem to be from trustworthy sources. Anti Phishing programmes, email filtration, and extensions for browsers are merely some of the tools and technologies which might detect phishing attempts based on URLs. These technologies identify possible phishing attacks using cutting-edge algorithms and machine learning approaches, warning users before they become victims.

List based Detection:

One of the conventional methods of detecting Phishing is list based detection. List based method includes both blacklist and whitelist ways of detection. Blacklist maintains a database of URLs, Ips and Domains that are suspicious and if one of the items of list is detected, they are marked as suspicious. Whitelist on the other hand is a form of maintaining a list of legitimate URLs and all the others are suspicious. Wang et al. [12], Han W et al. [13] and Jain AK et al. [14] used a whitelisting method for the detection of suspicious/Phishy URLs.

URL Based Detection Machine Learning Research papers:

Table 1 URL Based Machine Learning papers.

Research papers(authors)	Analysis of the Research paper		
	Method used	**Dataset**	**Results**
Sahingoz et al [1]	Use NLP based features, word vectors, and hybrid features, and then seven different machine learning algorithms are used to classify the URLs	A public dataset of 36,400 benign URLs and 37,175 phishing URLs	97.98% accuracy rate with RF
Rao et al. [2]	This technique proposes manually crafted URL features and TF-IDF based features and with the use of these features classifies the URLs by using random forest classifier	A public dataset of 85,409 benign URLs and 40,668 phishing URLs	TFIDF and hand-crafted features achieved accuracy of 94.26%
A. Lakshmanarao et al [7]	TF-IDF, Count and Hash vectorizer then different machine learning Algorithms (KNN, RF, DT, Logistic Regression)	Kaggle dataset with a large number of URLs (above 5,00000 URLs)	hash vectorizer and RF achieved 97.5% accuracy.

Deep Learning models

Through the analysis of various features including the URL, content, and images used in the attack, deep learning models have demonstrated promising results in the detection of phishing attacks. Phishing is a kind of cyberattack that involves pretending to be a trustworthy website or service to fool the user into disclosing sensitive information such as usernames, passwords, and credit card numbers.

Deep learning models, a type of machine learning, have the ability of discovering patterns in data by themselves without needing to be explicitly trained. To identify patterns and correlations that may be used to generate precise predictions on fresh data, these models can be trained on enormous datasets.

Convolutional neural networks (CNNs), recurrent neural networks (RNNs), and other deep learning models have all been applied to the identification of phishing attempts. To increase their accuracy and better recognise new and evolving assaults, these models can be trained on vast datasets of well known phishing attacks.

In addition to deep learning models, phishing detection can also be accomplished through feature engineering, rule-based systems, and ensemble models. Deep learning models, however, have proven to be more successful in identifying sophisticated and changing phishing attacks.

Overall, deep learning models have shown considerable potential in detecting phishing assaults, and as online criminals develop new techniques, their use in this area is likely to become even more crucial in the future.

URL Based Deep Learning Research papers:

Table 2 URL Based Deep Learning Research papers

Research papers(authors)	Analysis of the Research paper			
	Method used	**Encoding used**	**Dataset**	**Results**
Zheng et al. [5]	Proposed a new Highway Hierarchical Neural Network (HDP-CNN) to detect phishing URLs.	This method uses wordlevel embedding along with character-level	A private dataset contains 344,794 benign URLs and 71,556 phishing URLs	Accuracy at 98.30%, True positive rate (TPR) at 99.18% True negative rate (TNR) at 94.34%.
Le et al. [4]	This technique applies CNN networks to both characters and words of the URL string for malicious URL detection	characters and words encodings	A private dataset of 4,683,425 benign URLs and 9,366,850 malicious URLs	AUC score 99.2%

Vargas, J et al. [6]	compared a random forest classifier against recurrent neural networks based on a hand-crafted features method.	Character encoding for RNN	phishing URLs from Phishtank Legitimate URLs from Common Crawl	AUC statistic of 98.44% Accuracy : 93.47% recall : 93.28% precision : 93.63%
Wang et al [15]	proposed a fastphishing website detection method called precise phishing detection with recurrent convolutional neural networks (PDRCNN) that depends only on the URL of the website	Data preprocessing is based on word embedding	5,118,727 URL from PhishTank website 245,385 valid phishing	Accuracy of 97% AUC value of 99%

Email Content based Detection:

By examining the email's text and organizational structure as well as any links or attachments, content-based analysis can identify phishing emails.

Table 3 Email Content based Research Paper

Research papers(authors)	**Analysis of the Research paper**			
	Method used	**Encoding used**	**Dataset**	**Results**
Adwan Yasin et al [8] (Machine Learning Based approach)	The model applied the knowledge discovery procedures using five popular classification algorithms J48, Naïve Bayes, Support Vector Machine (SVM), Multi-Layer Perceptron and Random Forest.	TFIDF/COUNT VECTORIZOR	4598 spam emails from Nazario phishing corpus and 5940 ham emails from spam assassin project	99.1% accuracy was achieved using the Random Forest algorithm and 98.4% using J48,

Ioannis Agrafiotis et al [9]	phishing content classifier based on a recurrent neural network (RNN)	Word encoding	9 962 phishing emails Nazario phishing corpus and 10 000 emails from the Enron dataset	96.74 % accuracy 97.45 % precision 95.98 % Recall
Molly Dewis et al [10]	The technique involves the use of RNN model on both the textual email data and Numerical data.	TF-IDF	**TEXT:** Spam Assassin, Spam Classification for Basic NLP Spam Email **Numerical :** Spambase Email Spam Classification Dataset	average accuracy of 99% with the LSTM model for text-based datasets. average accuracy of 94% with the MLP model for numerical based datasets
YONG FANG et al [11]	The technique based on on recurrent convolutional neural networks (RCNN) model with multilevel vectors and attention mechanism,	WORD2VEC	7781 legitimate emails and 999 phishing emails from multiple sources, spam assassin, enron, Nazario corpus, wikileaks archive	overall accuracy reaches 99.848%

Chapter III – Design

Product Perspective

The Proposed system will give increased protection from phishing emails. The most conventional method against phishing is just awareness among users, such as: that they should not open files and links that are not verified and authorized or those not recognized by the users.
Some platforms use machine learning and deep learning approaches to tackle such problems. My aim is to work on the problem from a new perspective and develop a deep learning model for the task of phishing detection.

My system will help all users that wish to use my system, especially the naïve users who are not familiar with all the phishing techniques that the attacker uses to steal their valuable credentials.

Product Functions

The key features of Phisher.AI are highlighted below:

- A large dataset of real-world emails received by people in the past, which are equally divided into phishing emails and legit emails.
- A Deep Learning Model which is rigorously trained using the dataset to detect any potential phishing emails accurately and efficiently in the user's inbox.
- An email is sent as input to the ML model, which then analyses the email.
- After Analysing the email, the ML model outputs whether the email is a phishing email or a legit email. The model also informs, how confident in percentage it is on its result.

Interfaces

My prime aim of the project is to research the approaches to detect suspicious emails, and work on various ML and DL approaches.
One of the products that my research could provide is a browser extension. Following are the interfaces that may be used for the development.

User Interfaces

Browser contains an icon on the top bar, User may click on the icon to activate or trigger the process of email's text detection, sending the data to the PC server hosting the model, and after processing sending the data back to the user whether the email is suspicious or not.

Hardware Interfaces

This project aims to provide research that focuses on the results of phishing or spam detection in the emails. Out of many approaches that Iplan to carry out, one may deploy one of the models, based on appropriate metric results on a PC server and carry out communication via API request.

Software Interfaces

A browser extension may be used to read the email text for the further processing of the suspicious emails.

Communication Interfaces

The communication is done between the browser running the extension and the PC server requiring a working internet connection. cURL/ Axios may be used to send and receive the responses between the client and server.

Functional Requirements

Table 4 All Functional Requirements

FR	DESCRIPTION
FR1	User can click on the extension to send the email data to the server
FR2	The plugin should be able read the content of the mail
FR3	Server should be able to process the data
FR4	Server should Host the model and be able to perform processing on the data
FR5	The server should be able to classify the mail as spam or not
FR6	The server should be able to communicate back the results to the plugin
FR7	Plugin should be able to display the results to the user

Functional Requirements with Traceability information

Table 5 FR1 with traceability Information

Requirement ID	FR1		**Requirement Type**	Functional		**Use Case #**			
Status	***New***	X	***Agreed-to***	-	***Baselined***	-	***Rejected***	-	
Parent Requirement #	None								
Description	user can click on the plugin to send the email data to the server								
Rationale	Mail is sent to the plugin								
Source				**Source Document**	-				
Acceptance/Fit Criteria	Correct mail is sent to the plugin								
Dependencies									
Priority	***Essential***	X	***Conditional***	-	***Optional***	-			
Change History									

Table 6 FR2 with traceability Information

Requirement ID	FR2		**Requirement Type**		Functional		**Use Case #**		
Status	***New***	X	***Agreed-to***	-	***Baselined***	-	***Rejected***	-	
Parent Requirement #	FR1								
Description	The plugin should be able read the content of the mail								
Rationale	Plugin read all the content of the email without missing out any detail								
Source					**Source Document**	-			
Acceptance/Fit Criteria	The plugin read the content correctly								
Dependencies									
Priority	***Essential***	X	***Conditional***	-	***Optional***	-			
Change History									

Table 7 FR3 with traceability Information

Requirement ID	FR3		**Requirement Type**		Functional		**Use Case #**		
Status	***New***	X	***Agreed-to***	-	***Baselined***	-	***Rejected***	-	
Parent Requirement #	FR2								
Description	Server should be able to process the data								
Rationale	Server should process the data to generate the encoding to be fed to the model								
Source					**Source Document**	-			
Acceptance/Fit Criteria	All the data is correctly encoded								
Dependencies									
Priority	***Essential***	X	***Conditional***	-	***Optional***	-			
Change History									

Table 8 FR4 with traceability Information

Requirement ID	FR4		**Requirement Type**		Functional		**Use Case #**		
Status	***New***	X	***Agreed-to***	-	***Baselined***	-	***Rejected***	-	
Parent Requirement #	FR3								
Description	Server should Host the model and be able to perform processing on the data								
Rationale	All processed data should be passed to the model for processing								
Source					**Source Document**	-			
Acceptance/Fit Criteria	Whole email is fed to the model sequentially								
Dependencies									
Priority	***Essential***	X	***Conditional***	-	***Optional***	-			
Change History									

Table 9 FR5 with traceability Information

Requirement ID	FR5		**Requirement Type**		Functional		**Use Case #**		
Status	***New***	X	***Agreed-to***	-	***Baselined***	-	***Rejected***	-	
Parent Requirement #	FR4								
Description	The server should be able to classify the mail as spam or not								
Rationale	The classification should be correct. There should be as less as false-positives or false-negatives								
Source					**Source Document**	-			
Acceptance/Fit Criteria	The mail is correctly identified as spam or not								
Dependencies									
Priority	***Essential***	X	***Conditional***	-	***Optional***	-			
Change History									

Table 10 FR6 with traceability Information

Requirement ID	FR6		**Requirement Type**	Functional		**Use Case #**		
Status	***New***	X	***Agreed-to***	-	***Baselined***	-	***Rejected***	-
Parent Requirement #	FR5							
Description	The server should be able to communicate back the results to the plugin							
Rationale	The correct result as shown by the model should be sent to the plugin							
Source				**Source Document**	-			
Acceptance/Fit Criteria	Plugin receives the results							
Dependencies								
Priority	***Essential***	X	***Conditional***	-	***Optional***	-		
Change History								

Table 11 FR7 with traceability Information

Requirement ID	FR7		**Requirement Type**	Functional		**Use Case #**		
Status	***New***	X	***Agreed-to***	-	***Baselined***	-	***Rejected***	-
Parent Requirement #	FR6							
Description	Plugin should be able to display the results to the user							
Rationale	Plugin displays the correct results as received to the user							
Source				**Source Document**	-			
Acceptance/Fit Criteria	Users can see the results.							
Dependencies								
Priority	***Essential***	X	***Conditional***	-	***Optional***	-		
Change History								

Non-Functional Requirements

This section will discuss the non-functional requirements pertaining to this project and product scope. Non-Functional requirements focus on the quality attributes of the project such that they are evaluated on the criteria that to which degree these attributes are on par with the expectations of the user. The non-functional requirements are further elaborated through the division of evolution and execution. Execution qualities are those that are evaluated during runtime and Evolution qualities are those that are evaluated based on the flexibility of the system.

Execution Qualities Security

Security is a very important non-functional requirement for my project since the entire mission of my project is to improve my users' security by detecting phishing attempts. To meet this requirement, ensure all communication between the Phisher.AI extension and my servers is end-to end encrypted so that malicious interceptors cannot read sensitive emails of my users. It will also not store any plaintext emails on my servers.

Usability

This requires that my product is easy to learn and use. Phisher.AI meets this requirement by providing a simple and intuitive UI, which shows all information clearly. Once the extension has received the verdict from my server, the user can find out the result with a single glance at the extension icon. The extension icon changes colour depending on whether the email on the screen is a phishing email or a legit email is. To find out more about the email, the user can then click on the icon.

Reliability

This attribute refers to the likely period within which the system would run without suffering from complete system failure under specific conditions. To ensure reliability, preventative measures will be taken to reduce the possibility of my servers going down. Backup servers will also be developed in case a server does go down.

Compatibility

The compatibility of the system refers to how easily a product can be integrated with third-party systems. Since Phisher.AI is a web browser extension and there are a lot of different web browsers in the market, my extension must be compatible with all the major web browsers so that Ican help as many email users as possible. For the same reason, my extension should be able to detect phishing emails in any email client, be it Gmail, Outlook or any other.

Performance

The ideal performance of any product should be that it should work swiftly and smoothly and should be up-to-the standards of the industry. To provide optimal performance, my ML model is programmed as such that it runs efficiently. The servers also have powerful GPUs to help the model perform complex calculations quickly.

Evolution Qualities Testability

This refers to the degree of easiness of testing the product, the higher the testability the higher the chances of creating a robust and error-free product.

To ensure Phisher.AI's results are as accurate as possible, a large portion of my dataset is just for testing my ML model. And every time, It weak my ML model, it is automatically tested using the test data.

Maintainability

It is important to provide maintainability when creating a complex ML model like Phisher.AI. It helps make changes easy. Hence, it was made sure that all of the code uses meaningful variable and function names. And the code uses comments to explain important variables, functions, and classes.

Scalability

This refers to the ability of how far the product expands its processing capabilities upward and outward to increase the business growth of the product. Phisher.AI is a scalable service, and so it will continue to help the growing number of users to defend against phishing attacks.

Project Design and Process

Architecture Overview

In the figure below, it can be seen how multiple users can interact with the server via the internet.

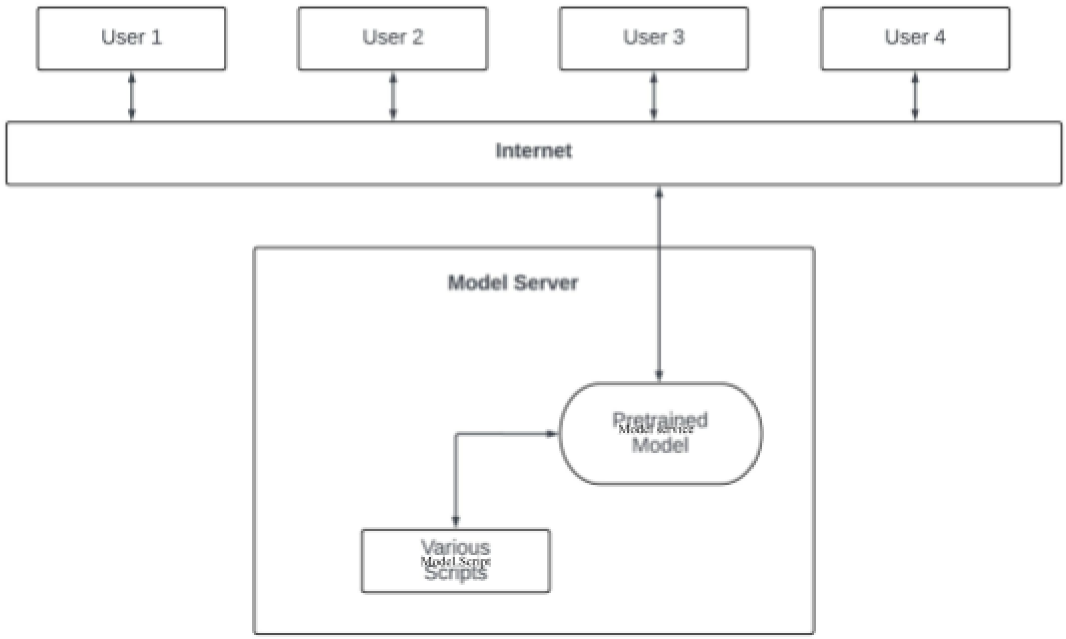

Figure 1 Architectural View of Extension

Use Case Diagram

Use case view shows that the design is complete in terms of functionality by incorporating use cases, actors, and their relationships.

In the figure below, it can be seen the user interacting with the web browser to use the functionality of the model.

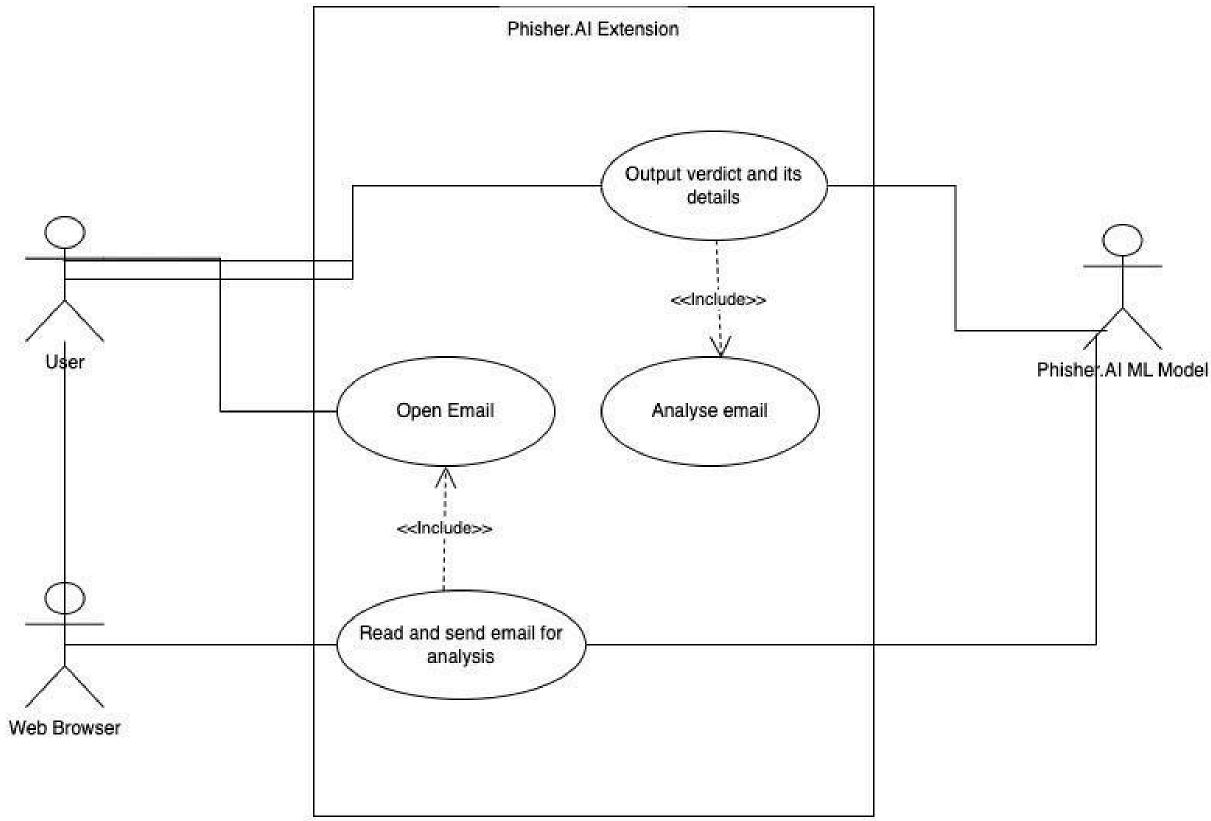

Figure 2 Use Case of Extension

Process View

Data Representation: chunks of **Xtn * 85** acts as input to the CNN (CONVOLUTIONAL NEURAL NETWORK), window slide (stride) equals to Xtn.

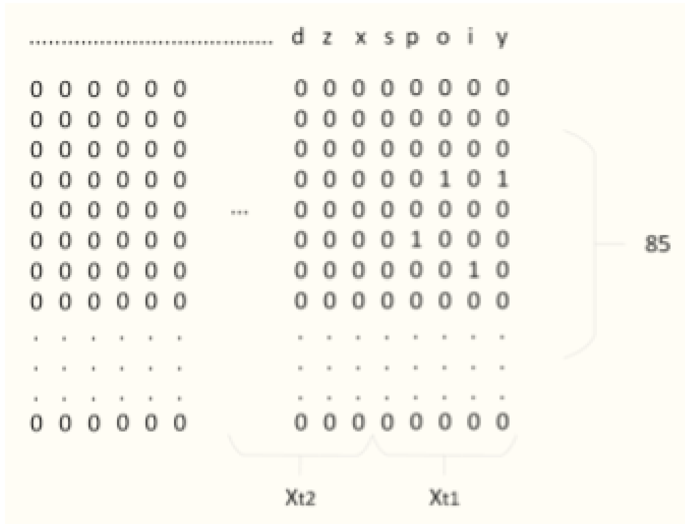

Figure 3 Data Representation (Character-wise Encoding)

CNN (CONVOLUTION NEURAL NETWORK)

AT T1: Xt1 * 85 would be fed into the neural network and the output is then provided to LSTM.

AT T2: Window slides to the left and **Xt2 * 85** would serve as the input of CNN and then to LSTM and so on until the end of email

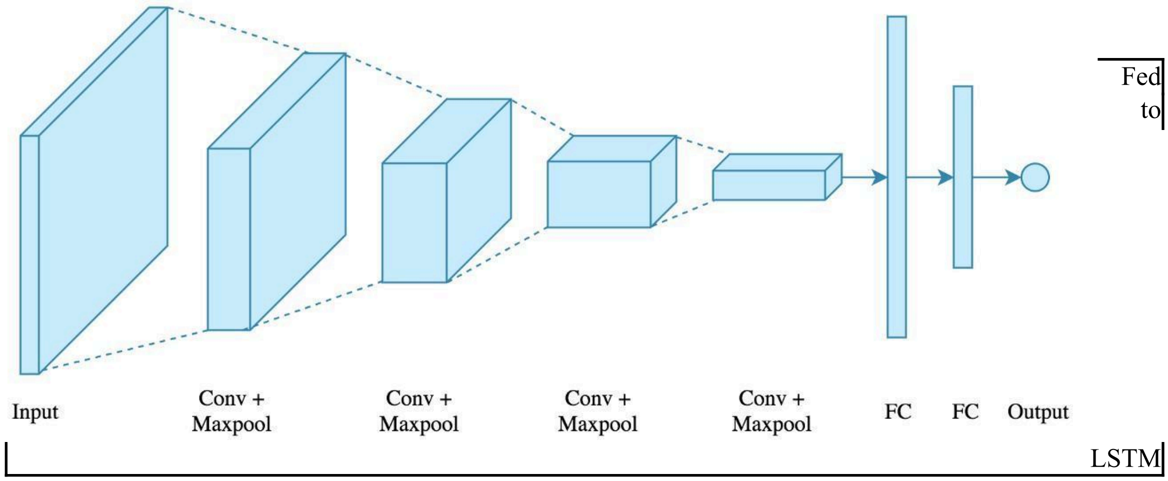

Figure 4 CNN (convolutional Neural Network)

https://medium.com/@mayankverma05032001/binary-classification-usingconvolution-neural-network-cnn-model-6e35cdf5bdbb

LSTM

For Sequential data, RNN is usually used. It can extract the sequential information from the input and to some extent contextual information as well. LSTM based RNN have long term dependencies as they are connected to the previous output and current input.

LSTM generates output as well as state to be passed to the next unit.

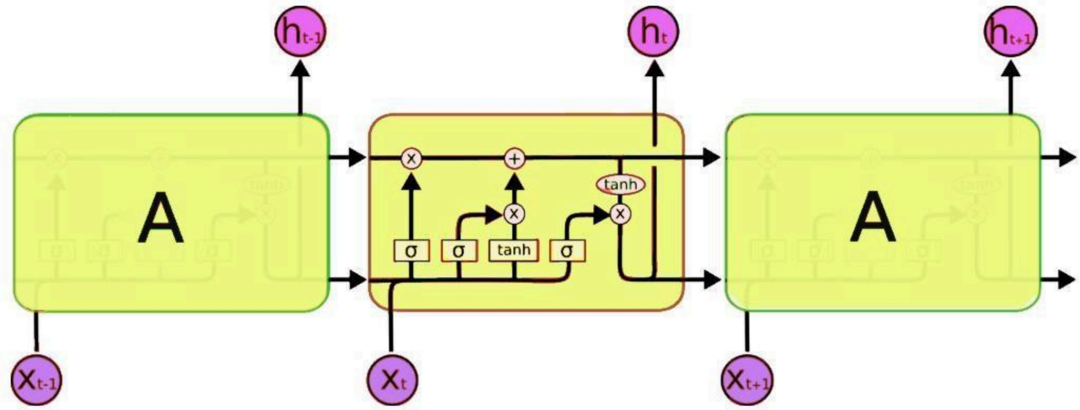

Figure 5 LSTM (Long Short-Term Memory)

https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deeplearning-introduction-to-lstm/

Logical View

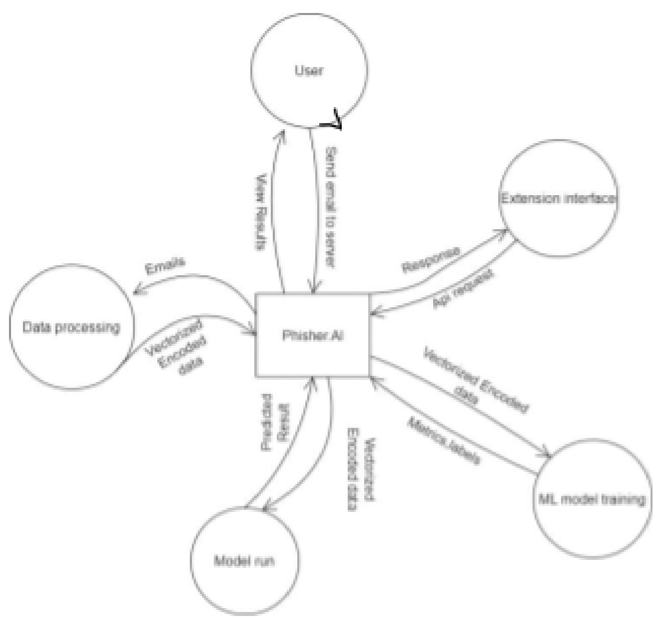

Figure 6 Logical View of PhisherAI

Development View

Development view gives the building block views of the system and describes static organization of the system module.

Component diagram

Figure shows the component diagram of generating results from the email.

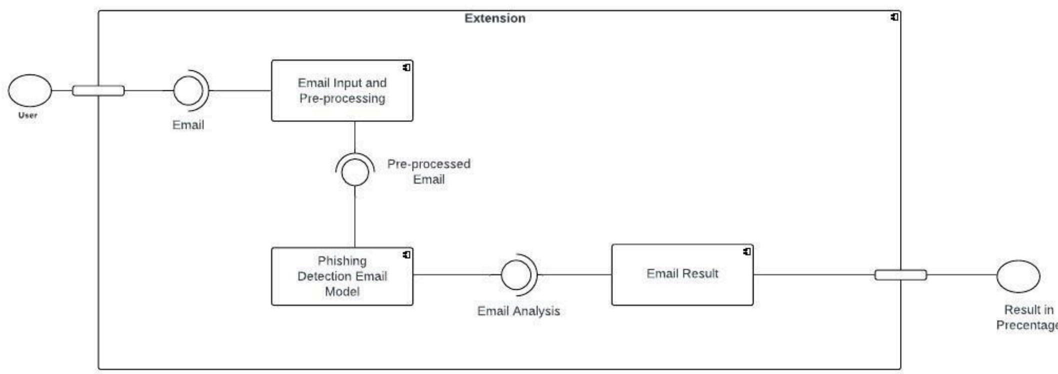

Figure 7 Component Diagram

Physical View

Figure represents the overview of the entire system.

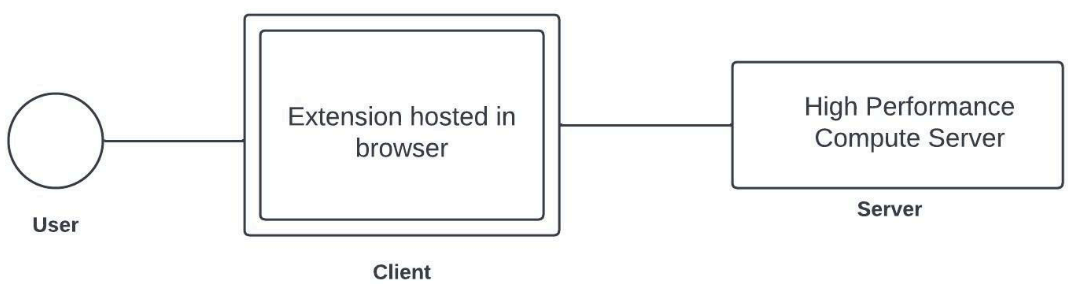

Figure 8 Physical View of Extension

User Interface

Following is the UI mockup design of the phiserAI extension.

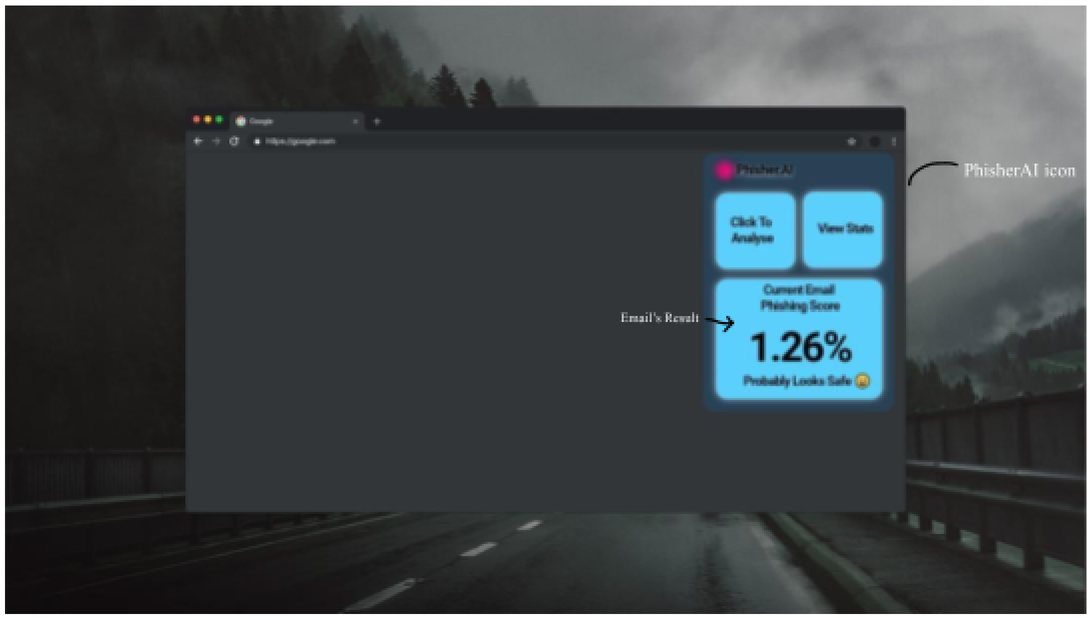

Figure 9 UI mockup Diagram

Chapter IV – Proposed Solution

Proposed Idea

Some of the previously done research was focused on detecting the phishing URLS, which required working on the dataset containing URLs that are either suspicious or not. Most of the datasets containing the URLs have become obsolete, as it's relatively easy to acquire a domain name, and attackers may register a new domain name. The aim of this project is to use the email content for the classification. Aim is to develop a deep learning model that would extract features on its own. A series of neural network algorithms such as RNNs, CNNs (Convolution Neural Network) and their combinations to generate a learning model is proposed. Moreover, an extension may be developed based on the most promising model.

Data Flow

For any machine learning related problem, an issue of how data should be structured to be fed to the model. There can be multiple ways to convert data into vector like representation, some of which are word2vec, TF-IDF, Count vectorized, one hot encoding. Previously, multiple vectorizers used to be deployed for traditional machine learning algorithms. For sequential data, one hot encoding is usually used.

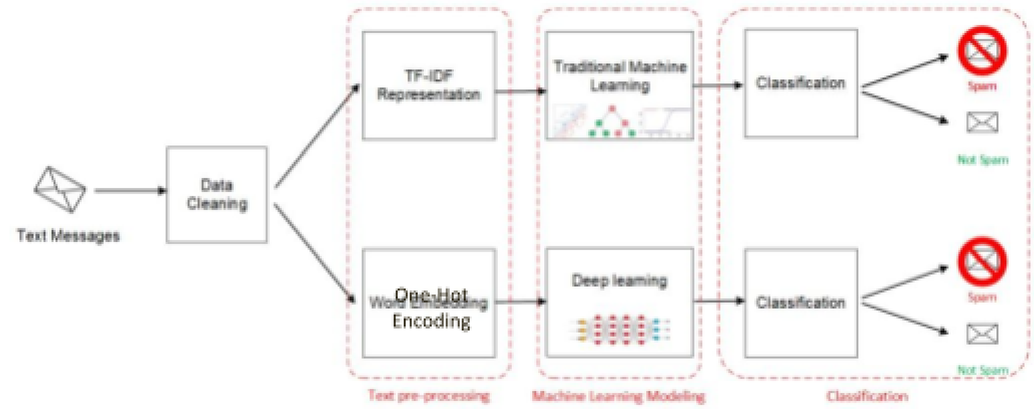

Figure 10 Data Flow Tradition ML vs DL

https://www.mdpi.com/1999-5903/12/9/156

ONE HOT ENCODING

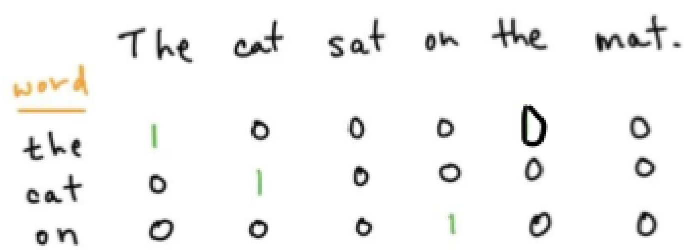

Figure 11 Word-wise one-hot encoding

https://www.analyticsvidhya.com/blog/2021/07/feature-extraction-and-embeddings-in-nlp-a-beginners-guide-to-understand-naturalla nguage-processing/

The above figure shows the word-wise encoding of a sentence with vocabulary of 6 words. As the vocabulary of words becomes larger the vector length represents each character as unique code becomes larger. So, in that case **character wise encoding** is used, as it will limit the vector length.

In this case a 98 length-vector is used, representing alphabets, numbers and characters used in a URL {A-Z, a-z, 0-9, @, %, ...}

Dataset Description

This project gave me a chance to work with datasets from two sources.

- The first one is a large phishing corpus that contains about **4600 real phishing emails** in a single *mbox* file from monkey.org.
- The second is a large database of **4600 real and non-phishing emails** obtained from the Enron Email Database

The link to the phishing corpus was retrieved from a GitHub repository that studied phishing emails and used external manually extracted features for the classification.

Project Flow

The deep learning model discussed here reads an email and then encodes it using character-level one-hot encoding. One-hot encoding is a methodology for turning categorical data, such as words or characters, into a numerical format that machine learning models can understand. In this scenario, each character in the content of the email is represented as a binary value vector, with a 1 in the position corresponding to the character's index and a 0 in the rest of the vector.

The one-hot encoded data is then passed through a convolutional neural network (CNN) layer, which is followed by a series of recurrent neural network (RNN) layers that include long short-term memory (LSTM), gated recurrent unit (GRU), attention, bidirectional LSTM (BiLSTM), and bidirectional GRU (BiGRU). The CNN layer extracts features from one-hot encoded input, while the RNN layers aid in capturing temporal dependencies in the email content.

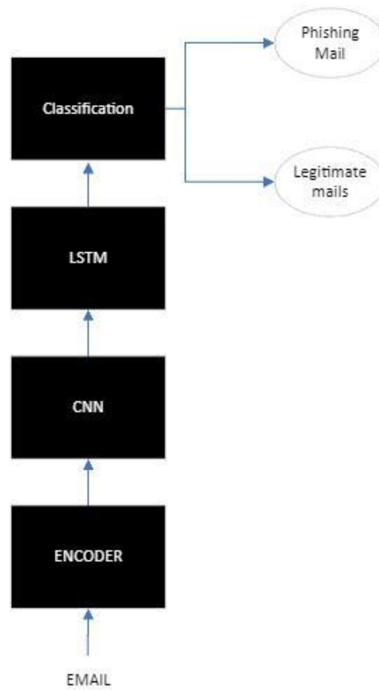

Figure 12 Process Flow

LSTM and GRU are two widely implemented RNNs that excel at processing sequential input. Attention methods direct the model's attention to the most relevant parts of the email content. In contrast, bidirectional RNNs allow the model to consider the context of the email by processing the text in both directions.

The model's output is a prediction percentage reflecting the likelihood of the email being a phishing email. A final dense layer determines this output by taking the output of the preceding layers and applying a sigmoid activation function, which results in a value between 0 and 1.

The approach presented here is very effective since it considers all an email's content, including the subject line and body text, instead of just certain features like the sender's email address or the URL included in the email. Furthermore, by combining CNN and RNN layers, the model can capture both the local and global context of the email content.

User Characteristics

This project is purely research based; however, Ido envision a full-fledged product based on the project. Here is a concept of the user characteristic of that product:

The user will have the Phisher.AI extension installed in their web browser of choice. The extension will detect an email open on screen in the browser and will inform the user whether the email is a phishing email or a legit email, and a percentage indicating the confidence level of the result.

Constraints

- The model’s learned context generalized to the provided dataset.
- Model can work with emails in the English language.
- There may be false positives and negatives.

- Time constraint of how long the model takes to analyse each email.
- Security constraint, communication between the Phisher.AI extension (client) and the Phisher.AI server should be encrypted to ensure confidentiality of the user's emails.

Assumptions and Dependencies

- The project is dependent on several different libraries for feature extraction, ML models, visualisation of results, reporting of results, etc.
- The speed of the ML model is dependent on the hardware. For intense calculations, the model might require high GPU (graphics processing units) power.
- The project is also network dependent for sending email content and results to and from the server.

Chapter V – Result and Discussion

The experimental findings and analysis of the described dataset using the suggested approaches in the sections that follow.

Several experiments were conducted using various settings, algorithms, and model combinations. For training and testing, the dataset was divided into two halves of 80% and 20% every experiment. Before using the learned model on the test dataset, the 80% datasets are further divided for validation during training. The AUC score and accuracy are then calculated for several thresholds after the trained model has been tested on 20% of the data.

Following are the number of experiments that were performed.

Table 12 Experiments Table

Experiments	Model Architecture	POOLING	Hyperparameters
1	CNN + LSTM	AVG/MAX	Convolutional Filters: 32, 64, 128; LSTM Units: 64, 128, 256
2	CNN + GRU	AVG/MAX	Convolutional Filters: 32, 64, 128; GRU Units: 64, 128, 256
3	CNN + BiLSTM	AVG/MAX	Convolutional Filters: 32, 64, 128; BiLSTM Units: 64, 128, 256
4	CNN + BiGRU	AVG/MAX	Convolutional Filters: 32, 64, 128; BiGRU Units: 64, 128, 256

After performing detailed experiments with all the architectures described in the above table the best results were achieved with the combination of convolution layers and GRUs. This model is a sequential neural network model used for some kind of classification or regression task. The model consists of several layers, including masking, convolutional layers, max pooling layers, recurrent layers (GRU), dropout layer, and dense layers.

Table 13 Model's specification

Encoding	RNN LAYERS	CNN LAYERS	Pooling
One-hot	2	3	Max pooling

The model achieved good performance on the task with the given hyperparameters. The training process involved running the model for 31 epochs, during which the model iteratively adjusted its parameters to minimize the loss function. The loss value of 0.3335 indicates that the model's predictions were not too far from the ground truth labels.

The accuracy of 0.8759 indicates that the model was able to correctly classify 87.59% of the input samples. The precision score of 0.8105 indicates that the model correctly identified 81.05% of the positive instances. The AUC score of 0.8759 indicates that the model's predictions are 87.59% better than random guesses.

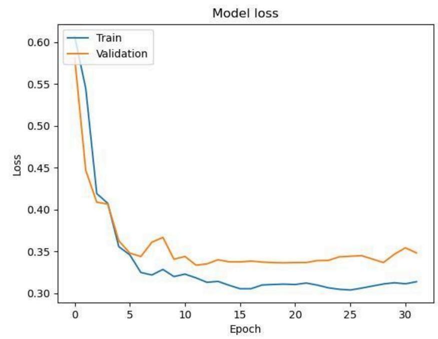

Figure 13 Best Model Loss Curve

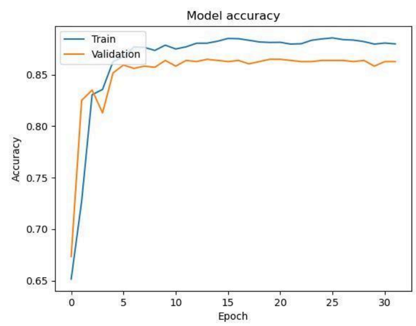

Figure 14 Best Model Accuracy Curve

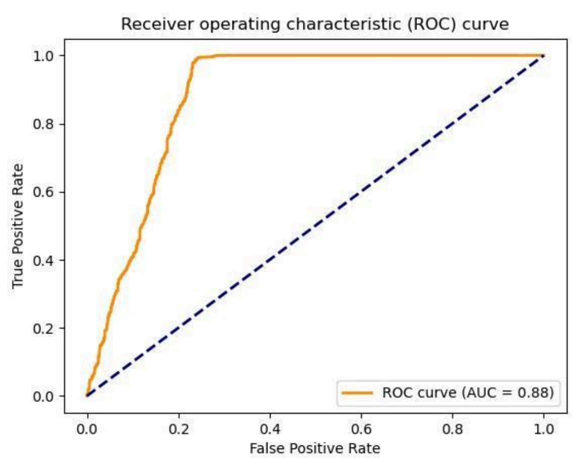

Figure 15 Best Model ROC Curve

The proposed model's architecture, which combines gated recurrent units (GRUs) and convolutional layers, was discovered to be the most successful in spotting and stopping phishing emails and SMS spam. Masking, convolutional, max pooling, recurrent (GRU), dropout, and dense layers are among the layers that make up the model architecture. Accuracy, precision, and AUC score were used to assess the model's effectiveness on the task.

The model's performance can be attributed to its capacity to use convolutional layers to extract important characteristics and GRU layers to capture the temporal dependencies in the input sequences. Additionally, the model can efficiently analyze the emails' content when given character-wise one-hot encoded vectors as input.

Chapter VI – Conclusion and Future work

This study examined the efficacy of character-level phishing email detection using a convolutional neural network (CNN) and recurrent neural network (RNN) model. According to the research, the character-wise one-hot encoded vectors used as input in the suggested CNN-RNN model can be quite effective at reducing the risks associated with phishing email assaults.

The outcomes of the tests showed that the proposed CNN-RNN model for phishing email detection has an accuracy of 87.59% and an AUC score of 88%. These findings show how the CNN-RNN model can successfully identify and stop phishing email attempts.

Overall, the research emphasizes the important part that deep learning models, like the CNN-RNN, can provide risk mitigation for cyberattacks. People and businesses can take proactive steps to guard against monetary losses, privacy violations, and identity theft by utilizing the power of deep learning to detect and block phishing email attacks at the character level.

While the study concentrated on the CNN-RNN model's performance in character-level phishing email detection, future research could examine other deep learning model types and assess how well they perform in identifying and thwarting different kinds of cyberattacks. Furthermore, additional research could be done into the creation of more reliable and precise deep learning models for identifying and preventing phishing email attacks at character level.

I investigated the possibility of using the suggested method to identify SMS spam in addition to phishing emails. A dataset of SMS texts was created, both authentic and spam, and used this dataset to test the CNN-RNN model. For SMS spam identification, the model was able to attain an accuracy of 98.54% and an AUC score of 99.62%. These findings suggest that the CNN-RNN model is also extremely effective at identifying SMS spam, a significant global cybersecurity risk that affects both individuals and businesses. The results indicate that the CNN-RNN model can be modified and applied to a variety of cybersecurity applications beyond the detection of phishing emails, and additional study may examine the usage of deep learning models for other cybersecurity applications.

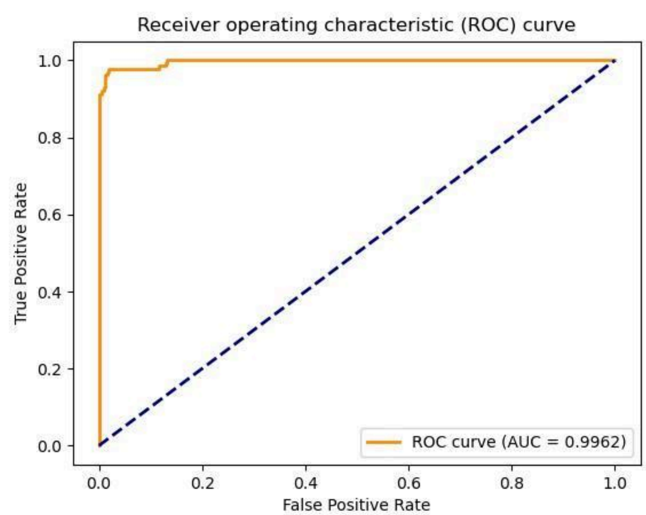

Figure 16 ROC curve for SMS Spam

GLOSSARY

Table 14 Term's Abbreviations

Name	Description
RF	Random Forest (ML model)
SVM	Support Vector Machine (ML Model)
DL	Deep learning
ML	Machine Learning
ROC	Receiver operating characteristic curve (Evaluation Metrics)
AUC	Area Under ROC curve (Evaluation Metrics)
TF-IDF	Term frequency-inverse document frequency (NLP Vectorization Technique)
UI/UX	User Interface/ User Experience
LSTM	Long Short-Term Memory (Deep learning model)
GRU	Gated Recurrent Neural Network (Deep learning model)
CNN	Convolutional Neural Network (Deep learning model)
RNN	Recurrent Neural Network (Deep learning model)
RcNN	Recursive Neural Network (Deep learning model)
FR	Functional Requirement

REFERENCES

1. Sahingoz OK, Buber E, Demir O, Diri B. Machine learning based phishing detection from URLs. *Expert Syst. Appl.* 2019
2. Rao RS, Vaishnavi T, Pais AR. Catch Phish: Detection of phishing websites by inspecting URLs. *J. Ambient. Intell. Humanized Compute.* 2019
3. Aljofey A, Jiang Q, Qu Q, Huang M, Niyigena J-P. An effective phishing detection model based on character level convolutional neural network from URL. *Electronics.* 2020
4. Le, H., Pham, Q., Sahoo, D., & Hoi, S.C.H. URL net: Learning a URL representation with deep learning for malicious URL detection. arXiv 2018,
5. Zheng, F., Yan Q., Victor C.M. Leung, F. Richard Yu, Ming Z. HDP-CNN: Highway deep pyramid convolution neural network combining word-level and character-level representations for phishing website detection, computers & security 2021
6. Vargas, J.; Gonzalez, F.A. Classifying Phishing URLs Using Recurrent Neural Networks. In Proceedings of the 2017 APWG Symposium on Electronic Crime Research (eCrime), Scottsdale,
7. A. Lakshmanarao, M. R. Babu, and M. M. Bala Krishna, "Malicious URL Detection using NLP, Machine Learning and FLASK," *2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, 2021
8. Yasin, Adwan & Abuhasan, Abdelmunem. (2016). An Intelligent Classification Model for Phishing Email Detection. International Journal of Network Security & Its Applications
9. Halgaš, I. Agrafiotis, and J. R. C. Nurse, “Catching the Phish:

Detecting Phishing Attacks Using Recurrent Neural Networks (RNNs)," *Information Security Applications*

10. M. Dewis and T. Viana, "Phish Responder: A Hybrid Machine Learning Approach to Detect Phishing and Spam Emails," *Applied System Innovation*, vol. 5, no. 4, p. 73, Jul. 2022.
11. Y. Fang, C. Zhang, C. Huang, L. Liu, and Y. Yang, "Phishing Email Detection Using Improved RCNN Model with Multilevel Vectors and Attention Mechanism," *IEEE Access*, vol. 7, pp. 56329–56340, 2019
12. Wang, Y., Agrawal, R., & Choi, B.Y. Light weight anti-phishing with user whitelisting in a web browser. in *region 5 conference, 2008 IEEE, IEEE*, 1–4
13. Han W, Cao Y, Bertino E, Yong J. Using automated individual whitelist to protect web digital identities. *Expert Syst. Appl.* 2012;**39**(15):11861–11869. doi: 10.1016/j.eswa.2012.02.020
14. Jain AK, Gupta BB. A novel approach to protect against phishing attacks at client side using auto-updated white-list. *EURASIP J. on Info. Security.* 2016; 9:1–11. doi: 10.1186/s13635-016-0034-3
15. Wang, W.; Zhang, F.; Luo, X.; Zhang, S. PDRCNN: Precise Phishing Detection with Recurrent Convolutional Neural Networks. *Secur. Commun. Netw.* 2019,

APPENDIX A

Project Code

Code enron to mbox

```
#!/usr/bin/env python

import mailbox
import sys
import email
import os import
glob import
shutil import
random



def maildir2mailbox(maildirname, mboxfilename):
    global emailsindex     global maxemails  # open the existing maildir
and the target mbox file     maildir = mailbox.Maildir(maildirname,
email.message_from_file)     mbox = mailbox.mbox(mboxfilename)

    # lock the mbox
mbox.lock()

    # iterate over messages in the maildir and add to the
mbox     for msg in maildir:         if emailsindex <
maxemails:
```

```
            mbox.add(msg)
emailsindex += 1          else:
            print("wrote " + str(maxemails) + " emails")
break

    # close and unlock
mbox.close()     maildir.close()



folders = [] emailsindex = 0 maxemails = 2279
outputfile = "enron-mbox"+str(maxemails)+".mbox"

# traverse root directory, and list directories as dirs and files as files
for root, dirs, files in os.walk("maildir"):     if files:
        folders.append(root)

for folder in folders:
  print("Processing " + folder)
os.makedirs(folder + "/cur")
os.makedirs(folder + "/new")   for file in
glob.glob(folder + "/[0-9]*_"):
    shutil.move(file, folder + "/cur")

os.makedirs("enron")

random.shuffle(folders) for
folder in folders:
    folder = folder.replace("\\cur", "")
```

```
    # path = "enron/" + folder.replace("\\", ".").replace("maildir", "enron")
path = "enron/" + outputfile     if emailsindex < maxemails:
print("Writing " + folder + " -> " + path)        maildir2mailbox(folder,
path)
```

code HTML to text

```
from urllib.request import urlopen
from bs4 import BeautifulSoup from
openpyxl import load_workbook

# url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
# html = urlopen(url).read()



#load excel file
workbook = load_workbook(filename="20051114.xlsx")
#open workbook sheet
= workbook.active

#modify the desired cell for
i in range(1,454):
column='E'+str(i)     html =
sheet[column]
print(html)     soup =
BeautifulSoup(html.value,
features="html.parser")
```

```
    # kill all script and style elements
for script in soup(["script", "style"]):
        script.extract()    # rip it out

    # get text     text =
soup.get_text()

    # break into lines and remove leading and trailing space on each
lines = (line.strip() for line in text.splitlines())     # break
multi-headlines into a line each     chunks = (phrase.strip() for line in
lines for phrase in line.split("  "))
    # drop blank lines     text = '\n'.join(chunk for
chunk in chunks if chunk)     sheet[column]=text
    #save the file workbook.save(filename="4.xlsx")
```

code mbox to csv

```
import mailbox import
csv

def mbox_to_csv(mbox_file, csv_file):     with
open(csv_file, 'w', newline='', encoding='utf-8') as f:
writer = csv.writer(f)        # Write the header row
writer.writerow(['Subject', 'From', 'To', 'Date', 'Content'])

        with open(mbox_file, 'rb') as mbox:
            mbox_reader = mailbox.mbox(mbox_file)
for message in mbox_reader:
```

```
            subject = message['subject'] if 'subject' in message else ''
from_email = message['from'] if 'from' in message else ''
to_email = message['to'] if 'to' in message else ''            date =
message['date'] if 'date' in message else ''            if
message.is_multipart():
             content = ''.join(str(part.get_payload(decode=True)) for part in
message.get_payload())            else:
             content = message.get_payload(decode=True)
writer.writerow([subject, from_email, to_email, date, content])

# Convert the .mbox file to .csv file
format mbox_file = "enron.mbox" csv_file
= "enron.csv" mbox_to_csv(mbox_file,
csv_file)
```

Code One Hot Encoding

```
#!/usr/bin/env python
# coding: utf-8

# # 1. Imports


# In[1]:



import pandas as pd import os import pickle import
numpy as np from matplotlib import pyplot as plt from
```

```
sklearn.utils import shuffle from sklearn.preprocessing
import LabelBinarizer as lbe

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv1D,
MaxPooling1D, LSTM


# # 2. Data Reading

# In[2]:




data_xls =
pd.read_excel('assets/data/unmod/dataPhish/1.xlsx',engine='openpyxl',index_
col=None)
data_xls1 =
pd.read_excel('assets/data/unmod/dataPhish/2.xlsx',engine='openpyxl',index_
col=None) data_xls = data_xls.append(data_xls1, ignore_index = True)

data_xls1 =
pd.read_excel('assets/data/unmod/dataPhish/3.xlsx',engine='openpyxl',index_
col=None) data_xls = data_xls.append(data_xls1, ignore_index = True)
```

```
data_xls1 =
pd.read_excel('assets/data/unmod/dataPhish/4.xlsx',engine='openpyxl',index_
col=None) data_xls = data_xls.append(data_xls1, ignore_index = True)

# data_xls = data_xls.sample(2200)


data_xls


# In[3]:




path="assets/data/unmod/dataValid" data_csv =
pd.read_csv(path+'/part1.csv',header=0) for i in
range(2,5):
   data_csv1 = pd.read_csv(path+'/part'+str(i)+'.csv',header=0)
data_csv = data_xls.append(data_csv1, ignore_index = True)
data_csv = data_csv.sample(4649) data_csv


# In[4]:



dataPhish = data_xls[["Subject","Content"]]
dataPhish['text'] = dataPhish['Subject'] +dataPhish['Content']
dataPhish = dataPhish[["text"]]
```

```
dataValid = data_csv[["Subject","Content"]] dataValid['text']
= dataValid['Subject'] +dataValid['Content'] dataValid =
dataValid[["text"]]

dataPhish["label"] = 1 dataValid["label"]
= 0


# In[5]:


dataPhish = dataPhish.dropna() dataValid
= dataValid.dropna()

dataValid['text'] = dataValid['text'].str.replace(r'\\[nt]+', ' ') dataPhish['text'] =
dataPhish['text'].str.replace(r'\\[nt]+', ' ')


# In[6]:


dataPhish


# In[7]:
```

```
lenghtsValid = [] for i in
dataValid[['text']].values:
lenghtsValid.append(len(str(i[0])))



def Average(lst):
   return sum(lst) / len(lst)

print(np.percentile(lenghtsValid, 90)) print(Average(lenghtsValid))



# In[8]:



lenghtsPhish = []
for i in dataPhish[['text']].values:
lenghtsPhish.append(len(str(i[0])))



def Average(lst):
   return sum(lst) / len(lst)

print(np.percentile(lenghtsPhish, 90)) print(Average(lenghtsPhish))



# In[9]:
```

```
combined = lenghtsValid+lenghtsPhish
print("Total : ",len(combined))
combined.sort(reverse=True)
print(np.percentile(combined, 90))
print(Average(combined)) print(max(combined))


# In[10]:


plt.plot(lenghtsValid) plt.plot(lenghtsPhish)
plt.show()


# # 3. New one hot encoding

# In[11]:


df = pd.concat([dataValid, dataPhish], ignore_index=True)


# In[19]:


df = df.sample(frac=1).reset_index(drop=True) df
```

In[23]:

```
label_counts = df['label'].value_counts()

# Plot the label frequencies as a bar plot fig,
ax = plt.subplots()
label_counts.plot.bar(ax=ax)

# Add the exact count of each label to the plot
for i, count in enumerate(label_counts):
ax.text(i, count+1, str(count), ha='center')

# Add axis labels and title to the plot
plt.xlabel('Label')
plt.ylabel('Frequency')
plt.title('Label Frequencies')

# Display the plot
plt.show()
```

In[44]:

```
num_files = 20  # The number of files to create batch_size
= len(df) // num_files
```

```
for i in range(num_files):     filename =
f'assets/data/data_{i}.csv'     start = i * batch_size     end =
(i + 1) * batch_size if i < num_files - 1 else len(df)
df_batch = df[start:end]     df_batch.to_csv(filename,
index=False)


# In[45]:


 classes = ['\n','\t','\r'] start =
32 for i in
range(start,start+95):
classes.append(chr(i))
print(len(classes))

lb=lbe()
lb.classes_=np.array(list(classes))

with open('assets/label/labels', 'wb') as f:
   pickle.dump(lb, f)


# In[52]:


filenames = []
```

```
for i in range(20):
    filenames.append('assets/data/data_{}.csv'.format(i)) filenames


# In[51]:



lb = None
# Open the pickle file in read binary mode
with open('assets/label/labels', 'rb') as f:
    # Load the contents of the file using pickle.load()
lb = pickle.load(f)

# Use the loaded data
lb


# In[53]:



max_len = 4000
num_classes = 98 result
= [] for filename in
filenames:
    # Load the CSV file into a Pandas DataFrame
df = pd.read_csv(filename)
```

```
   # Create a dataset from the DataFrame and apply the parse_data()
function     text = df['text']     label = df['label']

   # Create a dataset from the DataFrame and apply the parse_data()
function     payloads=df     payloads=payloads.fillna('').values
payloads=payloads[payloads.any(1)!='']
   payload = lb.transform(list(payloads[0,0]))

   if len(payloads[0,0]) < max_len:
      padding_vec = np.full((max_len-len(payloads[0,0]), num_classes), -1)
payload = np.concatenate((payload, padding_vec))     payload =
payload[:max_len]     x = payload     y = payloads[0,1]     for i in
range(1,len(df)):
      print(i,end=" ")        payload =
lb.transform(list(payloads[i,0]))        if
len(payloads[i,0]) < max_len:
         padding_vec = np.full((max_len-len(payloads[i,0]), num_classes), -1)
payload = np.concatenate((payload, padding_vec))

      payload = payload[:max_len]
x = np.append(x,payload,axis=0)
y = np.append(y,payloads[i,1])

   x =
np.resize(x,(len(df),max_len,num_classes))
with open(filename[:-4]+'_x.npy', 'wb') as f:
np.save(f, x)      with
open(filename[:-4]+'_y.npy', 'wb') as f:
```

```
np.save(f, y)
```

Code Training and testing

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:



import tensorflow as tf from tensorflow.keras
import metrics from tensorflow.keras.models
import Sequential
from tensorflow.keras.layers import AdditiveAttention ,Attention,Dense,
Input,dot,Activation,Lambda,Concatenate,concatenate,Dropout,Embedding,
Flatten, Conv1D, MaxPooling1D,AveragePooling1D
,LSTM,Masking,Bidirectional,GlobalAveragePooling1D,GRU, Permute, multiply

# from tensorflow.compat.v1.keras.layers import CuDNNGRU
from sklearn.preprocessing import LabelBinarizer as lbe from
tensorflow.keras import regularizers from
tensorflow.keras.models import Model

import numpy as np import
pandas as pd
```

```
import os from contextlib import
redirect_stdout import
matplotlib.pyplot as plt

import datetime
import pickle


# In[2]:


print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))


# # MODEL

# In[4]:


model = Sequential()


model.add(Masking(mask_value=-1, input_shape=(4000,98)))

model.add(Conv1D(filters=128, kernel_size=5, activation='relu'))

model.add(AveragePooling1D(pool_size=5))
```

```
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))

model.add(AveragePooling1D(pool_size=3))
# model.add(Conv1D(filters=128, kernel_size=2, activation='relu'))

# model.add(AveragePooling1D(pool_size=3))



model.add((GRU(units=256, return_sequences=True)))

model.add((GRU(units=128,   return_sequences=True)))

model.add((GRU(units=64)))

model.add(Dense(32, activation='relu'))

model.add(Dense(1, activation='sigmoid'))



optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)

model.compile(loss='binary_crossentropy', optimizer=optimizer,
metrics=['accuracy',tf.metrics.Precision(),tf.metrics.AUC()])



# In[9]:
```

```
model.summary()

# In[10]:


data_files = []
valFiles = []

noOfFiles=14

for i in range(noOfFiles):

data_files.append([['assets/data/data_{}_x.npy'.format(i)],['assets/data/data_{
}_y.npy'.format(i)]])


for i in range(noOfFiles,noOfFiles+2):

valFiles.append([['assets/data/data_{}_x.npy'.format(i)],['assets/data/data_{}
_y.npy'.format(i)]])


data_files,valFiles


# In[11]:
```

```
def data_generator(file_list, batch_size):
while True:         for file in file_list:
            x_train = np.load(file[0][0])            y_train
= np.load(file[1][0])            num_samples =
x_train.shape[0]            for i in range(0,
num_samples, batch_size):
                yield x_train[i:i+batch_size], y_train[i:i+batch_size]


# In[12]:



def lr_schedule(epoch):
         lr  =  0.001
if  epoch  >  10:
lr    =      0.0001
return lr

lr_callback = tf.keras.callbacks.LearningRateScheduler(lr_schedule)


# In[13]:



from keras.callbacks import EarlyStopping
```

```
batch_size = 128
num_samples = 455 #no of emails in one file
epch = 80

early_stopping = EarlyStopping(monitor='val_loss', patience=20,verbose=1)

history = model.fit(data_generator(data_files, batch_size),
steps_per_epoch=len(data_files)*num_samples/batch_size,validation_data=d
ata_generator(valFiles, batch_size),
validation_steps=len(valFiles)*num_samples/batch_size, epochs=epch,
callbacks=[early_stopping,lr_callback])

print("Epoch of early stop:", early_stopping.stopped_epoch)


# In[14]:


now = datetime.datetime.now() directory =
now.strftime("%Y-%m-%d_%H-%M-%S")

parent_dir = "assets/model/"

path = os.path.join(parent_dir, directory) os.mkdir(path)


# In[15]:
```

```
with open(path + '/modelsummary.txt', 'w') as f:
with redirect_stdout(f):        model.summary()

model.save(path + "/model.h5")


with open(path + '/history.pkl', 'wb') as f:
   pickle.dump(history.history, f) f.close()


# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy') plt.ylabel('Accuracy')
plt.xlabel('Epoch') plt.legend(['Train',
'Validation'], loc='upper left') plt.savefig(path +
'/acc.png') plt.show()
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss']) plt.title('Model
loss')
plt.ylabel('Loss') plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig(path + '/loss.png')
```

```
plt.show()


# In[16]:


def testGen(file_list, batch_size):
while True:         for file in
file_list:
            x_train = np.load(file[0][0])             y_train
= np.load(file[1][0])             num_samples =
x_train.shape[0]             for i in range(0,
num_samples, batch_size):
                yield x_train[i:i+batch_size], y_train[i:i+batch_size]


# In[17]:


data_files = []

for i in range(16,20):
data_files.append([['assets/data/data_{}_x.npy'.format(i)],['assets/data/data_{
}_y.npy'.format(i)]])

data_files
```

In[18]:

```
batch_size = 32 num_samples = 455 # emails in each file
result = model.evaluate(data_generator(data_files,
batch_size), steps=len(data_files)*num_samples/batch_size)
```

In[19]:

```
y_pred = [] y_true
= []

j = 0

for i in
data_files:     if
j==0:        j +=1
        x = np.load(i[0][0])
y_pred = model.predict(x)
y_true = np.load(i[1][0])
del(x)     x = np.load(i[0][0])

    y_pred=np.concatenate((y_pred, model.predict(x)), axis=0)
y_true = np.concatenate((y_true, np.load(i[1][0])), axis=0)
del(x)
```

```
# In[20]:

from sklearn.metrics import precision_recall_curve from
sklearn.metrics import roc_curve, auc

precision, recall, thresholds = precision_recall_curve(y_true, y_pred)

# Plot the precision-recall curve
plt.plot(recall, precision)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.savefig(path + '/precisionrecall.png')
plt.show()

# Compute the false positive rate, true positive rate, and
AUC fpr, tpr, thresholds = roc_curve(y_true, y_pred) roc_auc
= auc(fpr, tpr)
```

```
# Plot the ROC curve plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC
curve (AUC = %0.2f)'
% roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate') plt.ylabel('True Positive
Rate') plt.title('Receiver operating characteristic (ROC)
curve') plt.legend(loc="lower right") plt.savefig(path +
'/roc.png') plt.show()

with open(path + '/results.txt', 'w') as f:
   f.write(str(result) + " epochs : " +str(early_stopping.stopped_epoch))
f.close()
```

Extension model.py

```
import pickle import numpy as np
from keras.models import
load_model def
make_prediction(user_input, lb,
model, max_len=4000,
num_classes=98):
   user_input = lb.transform(list(user_input))
if len(user_input) < max_len:
      padding_vec = np.full((max_len - len(user_input), num_classes), -1)
user_input = np.concatenate((user_input, padding_vec))     user_input =
user_input[:max_len]     predictions =
model.predict(user_input.reshape((1, 4000, 98)))     return predictions
```

```
if __name__ == "__main__":     with
open("labels", "rb") as f:         lb =
pickle.load(f)     model =
load_model("model.h5")
print(make_prediction("Hello", lb, model))
```

Extension Server.py

```
from flask import Flask, jsonify, request import
pickle
from model import make_prediction
from keras.models import load_model
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

# Load the trained model and vectorizer
lb = pickle.load(open("labels", "rb"))
model = load_model("model.h5")



@app.route("/predict", methods=["GET"]) def
predict():
   user_input = request.args.get("q")     prediction =
make_prediction(user_input, lb, model)     score =
prediction[0][0]
```

```
    return jsonify({"prediction": float(score)})

@app.route("/predict", methods=["POST"]) def
predictP():
    # Get the input data from the request body
request_data = request.get_json(force=True)     user_input
= request_data['text']

    # Make the prediction and get the score
prediction = make_prediction(user_input, lb, model)
score = prediction[0][0]

    # Return the prediction score as JSON
return jsonify({"prediction": float(score)}) if
__name__ == "__main__":
app.run(debug=True, port=5000)
```