VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE-3501

INFORMATION SECURITY AUDIT AND ANALYSIS

FALL SEMESTER 2022-2023

DETECTING PHISHING WEBSITES USING RECURRENT NEURAL NETWORKS(RNN)

PROJECT REPORT BY:

KUSHAJ ARORA - 20BCI0047
ANANNYA CHULI- 20BCE2046
SHASHWAT KUMAR - 20BCE2818

SLOT-L55-56

SUBMITTED UNDER SUPERVISION OF:

PROF. CHANDRAMOHAN B

TABLE OF CONTENTS:

ABSTRACT

Security assaults on real sites to take clients' data, known as phishing assaults, have been expanding. This sort of assault doesn't simply influence people's or alternately associations' sites. Although a few location techniques for phishing sites have been proposed utilizing AI, profound learning, and different methodologies, their identification exactness should be upgraded. Genetic-algorithm based; a classifier approach is planned to be used for improving phishing website detection.

Use of SVM, Random Forest and Regression based Algorithms is lined up. But Recurrent Neural Network (RNN)—Long Short-Term Memory (LSTM) is one of the ML techniques that presents a solution for the complex real—time problems and has much higher accuracy than the other proposed solutions.

Internet and cloud technology improvements in recent years have significantly increased electronic trade, or consumer-to-consumer online transactions. The resources of an enterprise are harmed by this growth, which permits unauthorized access to sensitive information about users. One well-known assault that deceives users into accessing dangerous content and giving up their information is phishing.

Most phishing websites use the same website interface and universal resource location (URL) as the legitimate websites. There have been several recommended methods for identifying phishing websites, including blacklists, heuristics, etc. However, the number of victims is rising exponentially as a result of inadequate security technologies.

Internet users are more susceptible to phishing scams. Existing research demonstrates that the phishing detection system's performance is constrained. An intelligent method is required to safeguard users against cyber-attacks. In this paper, the author put forth a machine learning-based URL identification technique. To identify phishing URLs, arecurrent neural network technique is used. The results of the studies demonstrate that the suggested method performs more effectively in identifying malicious URLs than more current approaches.

INTRODUCTION

Phishing is a fraudulent technique that uses social and technological tricks to steal customer identification and financial credentials. Social media systems use spoofed emails from legitimate companies and agencies to enable users to use fake websites to divulge financial details like usernames and passwords. Hackers install malicious software on computers to steal credentials, often using systems to intercept username and passwords of consumers' online accounts. Phishers use multiple methods, including email, URLs, instant messages, forum postings, telephonecalls, and text messages to steal user information.

The structure of phishing content is similar to the original content and trick users to access the content in order to obtain their sensitive data. The primary objective of phishing is to gain certain personal information for financial gain or use of identity theft. Phishing attacks are causing severe economic damage around the world. For these reasons, phishing in modern society is highly urgent, challenging, andoverly critical. There have been several recent studies against phishing based on the characteristics of a domain, suchas website URLs, website content, incorporating both the website URLs and content, the source code of the website and the screenshot of the website.

However, there is a lack of useful anti phishing tools to detect malicious URL in an organization to protect its users. In the event of malicious code being implanted on the website, hackers may steal user information and install malware, which poses a serious risk to cybersecurity and user privacy. Malicious URLs on the Internet can be easily identified by analyzing it through Machine Learning (ML) technique.

Recurrent Neural Network (RNN)—Long Short-Term Memory (LSTM) is one of the ML techniques that presents a solution for the complex real—time problems. LSTM allow RNN to store inputs for a larger period.The combination of RNN and LSTM enables **to extract a lot of information from a minimum set of data**.

Therefore,it supports phishing detection system to identify a malicious site in a shorter duration. In comparison to most previous approaches, researchers focus on identifying malicious URLs from the massive set of URLs. Therefore, the study proposes Recurrent Neural Network (RNN) based URL detection approach. The objectives of the studyare as follows:

1. To develop a novel approach to detect malicious URL and alert users.

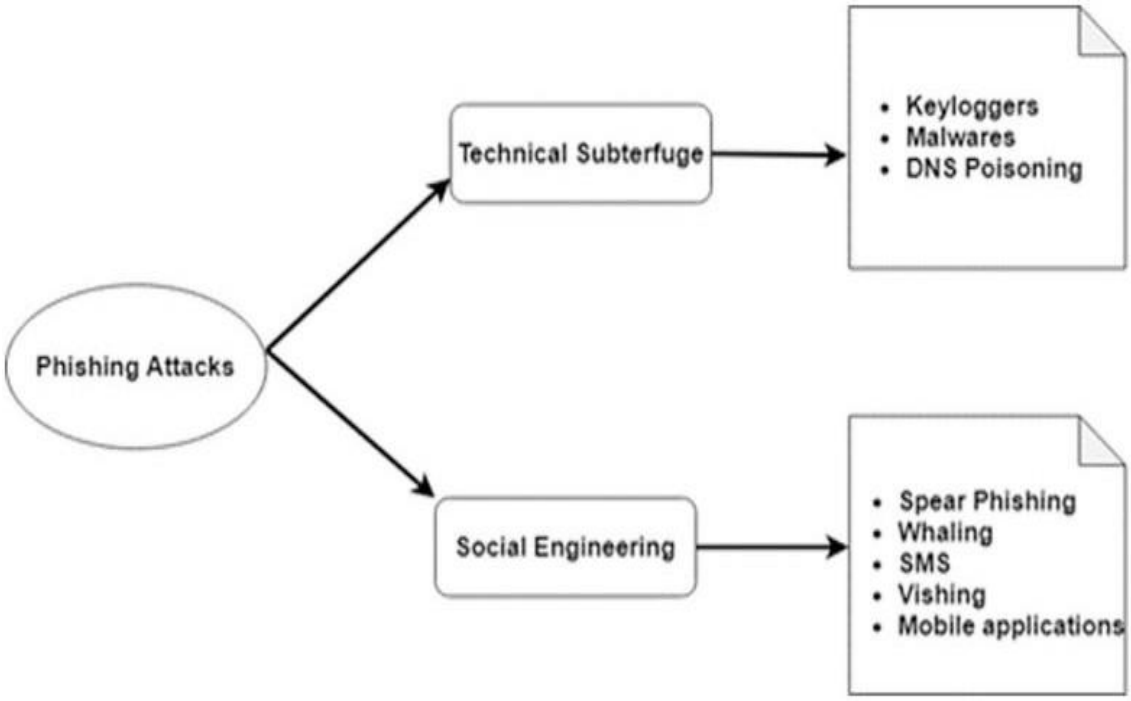
2. To apply ML techniques in the proposed approach in order to analyze the real time URLs and produce effectiveresults.

3. To implement the concept of RNN, which is a familiar ML technique that has the capability to handle hugeamount of data.

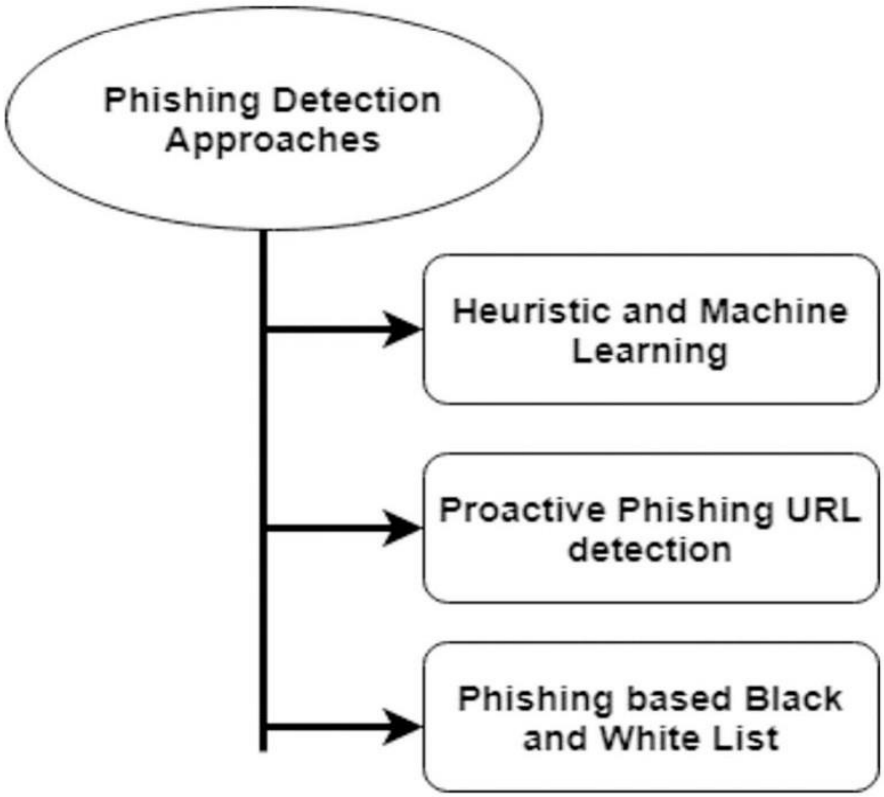
Phishing detection schemes which detect phishing on the server side are better than phishing preventionstrategies and user training systems. These systems can be used either via a web browser on the client or through specific host-site software. The figure presents the classification of Phishing detection approaches. Heuristic and ML based approach is based on supervised and unsupervised learning techniques. It requires features or labels for learning an environment to make a prediction. Proactive phishing URL detection issimilar to ML approach. However, URLs are processed and support a system to predict a URL as a legitimate or malicious. The performance of the detection systems is calculated according to the following:

- **Number of True Positives (TP):** The total number of malicious websites.
- **Number of True Negatives (TN):** The total number of legitimate websites.
- **Number of False Positives (FP):** The total number of incorrect predictions of legitimate websites as amalicious website.
- **Number of False Negatives (FN):** The total number of incorrect predictions of malicious websites as a legitimate website.

Multiple forms of phishing attacks:



Anti-Phishing Approaches:



Classification of phishing attack techniques Phishing websites are challenging to an organization and individual due to its similarities with the legitimate websites. Phishing attacks are categorized according to Phisher's mechanism for trapping alleged users. Several forms of these attacks are keyloggers, DNS toxicity, etc.

The initiation processes in social engineering include online blogs, short message services (SMS), social media platforms that use web 2.0 services, such as Facebook and Twitter, file-sharing services for peers, Voiceover IP (VoIP) systems where the attackers use caller spoofing IDs. Social engineering attacks include Spear phishing, Whaling, SMS, Vishing, and mobile applications.

Each form of phishing has a little difference in how the process is carried out in order to defraud the unsuspecting consumer. E-mail phishing attacks occur when an attacker sends an e-mail with a link topotential users to direct them to phishing websites.

Technical subterfuge refers to the attacks include Keylogging, DNS poisoning, and Malwares. In these attacks, attacker intends to gain the access through a tool / technique. Attackers focus on the group of people or anorganization and trick them to use the phishing URL.

Therefore, Phishing can be alluded to as a modernized wholesale fraud, which takes the advantage of human instinct and the Internet to trap a huge number of individuals and take a lot of cash. It has been logical that in the most recent couple of years phishing assaults have quickly become a genuine danger to worldwide security. The primary endeavour of these is to utilize the weaknesses present in the framework, which might be either specialized or because of client absence of knowledge.

Phishing is a genuine cybercrime and generally regular of all. As indicated by measurements, 1 in each 99 messages is a phishing assault. In 2021 83% of individuals got phishing assaults overall bringing about a scope of interruptions and harms. This incorporates diminished profitability by 67%, loss of legitimacy information by 54%, and harm to properties by half. The underline or more insights plainly shows that phishing is a difficult issue in various zones. With creating phishing methods, AI is the weapon which can diminish this assault generally. Utilizing AI, we will prepare our model to the informational collection containing the highlights of the phishing sites.

Literature Survey

JOURNAL	PROBLEM STATEMENT	TECHNIQUE USED	ADVANTAGES OF TECHNIQUE	COMPARISON STUDY	LIMITATIONS
Malicious web pages detection using feature selection techniques and machine learning	In recent years, researchers have provided significant solutions to detect malicious web pages, still there are many open issues. This paper proposes a methodology for the effective detection of malicious web pages using feature selection methods and machine learning	Our methodology consists of three modules: feature selection, training and classification. To evaluate our methodology, six feature selection methods and eight supervised machine learning classifiers are used. Experiments are performed on the balanced binary dataset	It is found that by using feature selection methods, the classifiers achieved significant detection accuracy of 94-99% and above, error-rate of 0.19-5.55%, FPR of 0.006- 0.094, FNR of 0.000-0.013 with minimum system overhead which is very high.	After the comparison of the techniques used feature selection classifiers had the highest precision and accuracy. ranging up to 99% and above and very commendable error rate	For instance, time taken bythis feature selection and classification is high and turns out to have a negative impact on the accuracy and the system.
Intelligent phishing website detection using random forest classifier	Phishing is defined as mimicking a creditable company's website aiming to take private information of a user. Data mining is a promising technique used to detect phishing attacks. In this paper, an intelligent system to detect phishing attacks is presented.	We used different data mining techniques to decide categories of websites: legitimate or phishing. Different classifiers were used in order to construct an accurate intelligent system for phishing website detection. Classification accuracy, area under receiver operating characteristic (ROC) curves (AUC) and Measure is used to evaluate the performance of the data mining techniques	advantages include ability to classify patterns on which they have not been trained and high tolerance for noisy data. In addition to that, due to their parallel nature, parallelization techniques can speed up the computational process and rule extraction can be done through certain techniques.	The performance of the proposed RF classifier is rather high in terms of classification accuracy Measure and AUC. Furthermore, our results showed that RF is faster, robust and more accurate than the other classifiers. Random forest's runtime is quite fast, and it is able to detect phishing websites in comparison to the other classifiers.	The limitation of the proposed model is that the induction of rules needs a large number of rules.
Phishing Attacks and Websites Classification Using Machine Learning and Multiple Datasets–	Phishing assaults are widely recognized sort of digital assaults used to get delicate data and have been influencing people just as associations across the globe. Different strategies have been proposed to recognize phishing assaults explicitly, sending machine insight as of late.	The experiments are designed by utilizing different ML and data analytics libraries including Scikit Learn, Keras, NumPy, and Pandas. Five ML algorithms namely, decision tree, random forest, Naïve Bayes, K-Nearest Neighbours, and artificial neural networks were employed.	It was basically a comparison of different data-mining techniques for prediction. Thus, the techniques used helped find the best algorithm to predict phishing attacks. The best algorithm is Neural Networks.	After the comparison of the techniques used Random Forest and Artificial Neural Networks had the highest precision and accuracy.	For instance, it did not investigate the formation of a dataset consisting of the integrated features identified in this study. It did not use the ML techniques to classify the more complex problems (i.e., adversarial attacks) in this domain. It needs to be further consolidated as it is a very vague study.

Particle Swarm Optimization-Based Feature Weighting for Improving Intelligent Phishing Website Detection	Throughout the most recent couple of years, web phishing assaults have been continually advancing making clients lose trust in web-based business and online administrations. Different instruments and frameworks dependent on a boycott of phishing sites are applied to recognize the phishing sites. Tragically, the quick development of innovation has prompted the bringing into the world of more complex techniques when building sites to draw in clients.	A methodology of the proposed intelligent phishing website detection based on PSO-based feature weighting. The methodology consists of two main phases: training phase and detection phase. In the proposed PSO-based feature weighting, the best weights of website features are heuristically generated using PSO to maximize the performance of phishing website detection.	Simple concept, easy implementation, robustness to control parameters, and computational efficiency when compared with mathematical algorithm and other heuristic optimization techniques.	After using PSO based feature weighting, accuracy of almost each classifier technique increased.	Faster and improved version of PSO can be used to speed up the performance of the proposed PSO-based feature weighting. Furthermore, other phishing websites datasets should be used to validate and evaluate the proposed PSO-based feature weighting. Lastly, the proposed PSO-based feature weighting was used to enhance some classical machine learning classifiers.
AAI Meta-Learners and Extra-Trees Algorithm for the Detection of Phishing Websites –	It covers various aspects of computer and network security including Intrusion Detection System (IDS), Antivirus, Phishing etc. Phishing is a nefarious cyber-attack plaguing the digital world with a direct impact on the physical world. Phishing is now a well-known subject-matter and the effect of successfully conducting phishing attacks is known to be disastrous.	This study considers the phishing detection problem as an AI-based classification problem wherein the result of the decision-making phase leads to detecting if a given website is either a legitimate or a phishing website. The selected meta-learners who are Bagging, AdaBoost, Rotation Forest, and Logit Boost, aswell as one base learner (i.e., the Extra-tree algorithm), were the selected algorithms to be used in this study.	Meta-learning allows machine learning systems to benefit from their repetitive application. If a learning system fails to perform efficiently, one would expect the learning mechanism itself to adapt in case the same task is presented again.	ROFBET, LBET, BET, ABET algorithms were compared and LBET had the highest accuracy of all AI Meta Learners	The development of interpretable AI methods remains as a predominant concern in the AI community and the continuous contribution towards implementing interpretable AI models is essential.

Phishing page detection via learning classifiers from page layout feature	The web technology has become the cornerstone of a wide range of platforms, such as mobile services and smart Internet-of-things (IoT) systems. In such platforms, users' data are aggregated to a cloud-based platform, where web applications are used as a key interface to access and configure user data.	We prototyped our approach and evaluated it based on four learning classifiers, namely, Support Vector Machine (SVM), Decision Tree, AdaBoost, and Random Forest. Evaluation used more than 490 phishing web pages from phishtank.com that mimic 46 targets pages, from which we extracted over 20,000 testing samples.	More effective in high dimensional spaces. SVM is effective in cases where the number of dimensions is greater than the number of samples. SVM is relatively memory efficient.	ROFBET, LBET, BET, ABET algorithms were compared and LBET had the highest accuracy of all AI Meta Learners.	Among these four classifiers, Random Forest performs the best by considering all the four metrics. All the classifiers show more than 93% accuracy and more than 84% F1, which demonstrates that our approach can make an effective detection in phishing websites.
A Stacked Deep Learning Approach for IoT Cyber attack Detection using five pretrained residual networks (Res Nets)	Phishing assaults are the most widely recognized sort of digital assaults used to get delicate data and have been influencing people just as associations across the globe. Different strategies have been proposed to recognize phishing assaults explicitly, sending machine insight as of late.	Stacked Generalization Ensemble and CNN networks- a technique that combines multiple base-level classification models via a meta classifier or meta learner. CNN is a type of neural network model which allows us to extract higher representations for the image content. We stacked five deep Res Net models, and the outputs of these models are connected to a new model that takes their output as new training data.	CNN is composed of multiple processing layers, such as convolutional, subsampling, and fully connected layers therefore CNN is making major advances in solving very complicated issues that cannot be solved by other algorithms. A Res Net Block has a special underlying mapping and the "identity shortcut connection" feature, which skips one or more layers.	The average accuracy of this method was 97.5%, which is better than that of the other methods. This method achieved the best performance in terms of accuracy on the N-BaIoT dataset. The accuracy of our method and the accuracy of the second-best method (i.e., random forests) are 100% and 99.9991%, respectively.	This needs to investigate transfer learning and several other state-of-the-art pretrained models to improve the performance of IoT IDSs in terms of accuracy and detection time also integrate other heterogeneous IoT environments to prove that IoT network traffic generated by cybercriminals are similar, and a general IDS can be deployed in heterogeneous IoT environments.

Phishing Website Detection using Machine Learning Algorithms using deep learning techniques by Rishikesh Mahajan, Irfan Siddavatam (2018)	Phishing attack is a simplest way to obtain sensitive information from innocent users. Aim of the phishers is to acquire critical information like username, password and bank account details.	Decision Tree Algorithm-Decision tree creates training model which is used to predict target value or class in tree representation each internal node of the tree belongs to attribute and each leaf node of the tree belongs to class label, Random Forest Algorithm-tree in forest predicts the target value and then algorithm will calculate the votes for each predicted target, Support Vector Machine Algorithm (SVM) is used for classification and detection.	Decision tree algorithm is easy to understand and also easy to implement. Random forest algorithm is one of the most powerful algorithms in machine learning technology and it is based on the concept of decision tree algorithm. Provides more accuracy for prediction of phishing websites.	: Result shows that Random forest algorithm gives better detection accuracy which is 97.14 with lowest false negative rate than decision tree and support vector machine algorithms. Detection accuracy increases when 90% of data used as training dataset and random forest detection accuracy is greater than other two classifiers.	Implemented to detect phishing websites more accurately, for which random forest algorithm of machine learning technology and blacklist method should be used.
Phishing Website Detection based on Supervised Machine Learning with Wrapper Features Selection by Waleed Ali	The problem of Web phishing attacks has grown considerably in recent years and phishing is considered as one of the most dangerous Web-crimes.	Back-Propagation Neural Network (BPNN), Radial Basis Function Network (RBFN), Support Vector Machine (SVM, Naïve Bayes Classifier (NB), Decision Tree (C4.5) and Random Forest (RF), K-Nearest Neighbor (kNN).	BPNNs are the most well known algorithms in neural network models. Architecture of RBFN consists of a three-layer feedback network: an input layer, a hidden layer and an output layer. SVM is used to transform the data into a high dimension to use linear discriminant functions. KNNis used to transform the datainto a high-dimension to uselinear discriminant functions	Supervised machine learning classifiers with the PCA features selection method achieved the worst TPR, TNR, and GM for the phishing websites dataset. The results showed that BPNN produces the highest accuracy (88.7%) followed by KNN (87%) and SVM (80%).	The experimental results showed that BPNN, kNN and RF achieved the best CCR while RBFN and NB achieved the worst CCR for detecting the phishing websites. The machine learning classifiers based on wrapper-based features selection accomplished the best performance while these classifiers with PCA features selection method achieved the worst performance in terms of CCR, TPR, TNR, and GM.

Hybrid intelligent phishing website prediction using deep neural networks with genetic algorithm-based feature selection and weighting (2019)	As web phishing attack is one of the most serious web security problems, where phishers can steal very detailed personal information of the users. So, in this paper, Prediction of phishing websites is done using deep neural networks (DNNs) with evolutionary algorithm- based feature selection and to enhance the prediction using some weighting methods. The features extracted by GA are used to train DNNs to predict more accurately.	Firstly, preprocessing is done which involves 2 steps-feature extraction, preparation of training data set. Genetic algorithm is used to find the most important features for enhancing website detection. DNNs are trained using the training data set with the best features extracted. Deep neural networks (DNNs) are used with evolutionary algorithm-based feature selection and weighting method to get a better prediction. Evaluated using four common measures: classification accuracy, sensitivity, specificity and GM	Provides more accuracy for prediction of phishing websites, in terms of specificity, the performance is higher of DNN, the technique was able to predict both legitimate and phishing websites, using GA, it hasprovided more accurate results.	It shows the performance of DNN compared to other ML algorithms. The results showed that DNN produces highest accuracy (88.7%) followed by KNN (87%) and SVM (80%)	The proposed hybrid phishing websites prediction approaches should be tested and evaluated using other phishing websites datasets with many websites features, feature selection and weighting using GA requires a longer time, other faster EAs could be utilized as a feature selection technique with DNNs to produce a more efficient performance of the phishing websites detection/
Efficient deep learning techniques for the detection of phishing websites	Phishing is a fraudulent practice and a form of cyber-attack designed and executed with the sole purpose of gathering sensitive information by masquerading the genuine websites. Phishers fool users by replicating the original and genuine contents to reveal personal information such as security number, credit card number, password, etc.	Several anti-phishing approaches have already been developed, including blacklist- or whitelist-based, heuristic feature-based, and visual-similarity-based methods. Therefore, in the proposed technique we have tried to reduce that. Our technique includes:-(a) Deep Neural Network (DNN), (b) Long Short-Term Memory (LSTM) used for classifying, prediction based on time series data, (c) Convolution Neural Network (CNN) have been proposed.	The proposed technique achieves an accuracy of 99.52% for DNN, 99.57% for LSTM and 99.43% for CNN, the proposed techniques utilize only one third-party service feature, thus making it more robust to failure and increases the speed of phishing detection.	Accuracy increases to 99.5% after using DNN with LSTM and CNN.	Linear layers in LSTM require large amounts of memory bandwidth to be computed, CNN requires a large dataset which is not used in this paper, due to several layers in CNN the training process takes a lot of time.

PROBLEM STATEMENT

Security researchers are currently really concerned about phishing since it is so simple to develop a phoney website that closely resembles an authentic website. Even though professionals can spot bogus websites, not all users can, leaving them vulnerable to phishing attacks. The simplest method of obtaining sensitive information from unwitting people is through a phishing attack.

The goal of phishers is to obtain crucial data, such as username, password, and bank account information. Detecting dangerous websites has significantly improved recently, however there are still a lot of problems that need to be solved. Security specialists are now searching for reliable and consistent detection methods to identify phishing websites.

Hence the purpose of this project is to detect phishing URLs and narrow down the best machine learning algorithm by comparing accuracy rate, false positives, and false negatives.

Linear layers in LSTM require large amounts of memory bandwidth to be computed, CNN requires a large datasetwhich is not used in this paper, due to several layers in CNN the training process takes a lot of time.

EXISTING SOLUTIONS

- Six feature selection approaches and eight supervised machine learning classifiers are used in tests on the balanced binary dataset to assess this methodology.
- In order to create a precise intelligent system for phishing website identification, various classifiers were applied. The effectiveness of the data mining approaches is assessed using classification accuracy, area under the ROC curves (AUC), and the F-measure.
- Three distinct machine learning approaches were used to compare the performance of two feature selection methods,

  Feature Selection by Omitting Redundant and Feature Selection: Random Forest (RF) tree, Multilayer Perceptron (MLP), and Naive Bayes (NB).
- Utilizing different ML and data analytics libraries including Scikit Learn, Keras, NumPy, and Pandas and five ML

  algorithms namely, decision tree, random forest, Naïve Bayes, K-Nearest Neighbours, and artificial neural networksto find the best algorithm to predict phishing attacks.
- To detect malicious URLs, character-level multi-spatial deep learning model, convolutional neural networks (CNN)

  have been used to improve detection performance. The proposed model was also integrated into a single board computer resulting in an energy-saving and efficient phishing website sensor to examine the feasibility of using resource- constrained computing devices to provide a phishing website detection sensor, as far as we know.

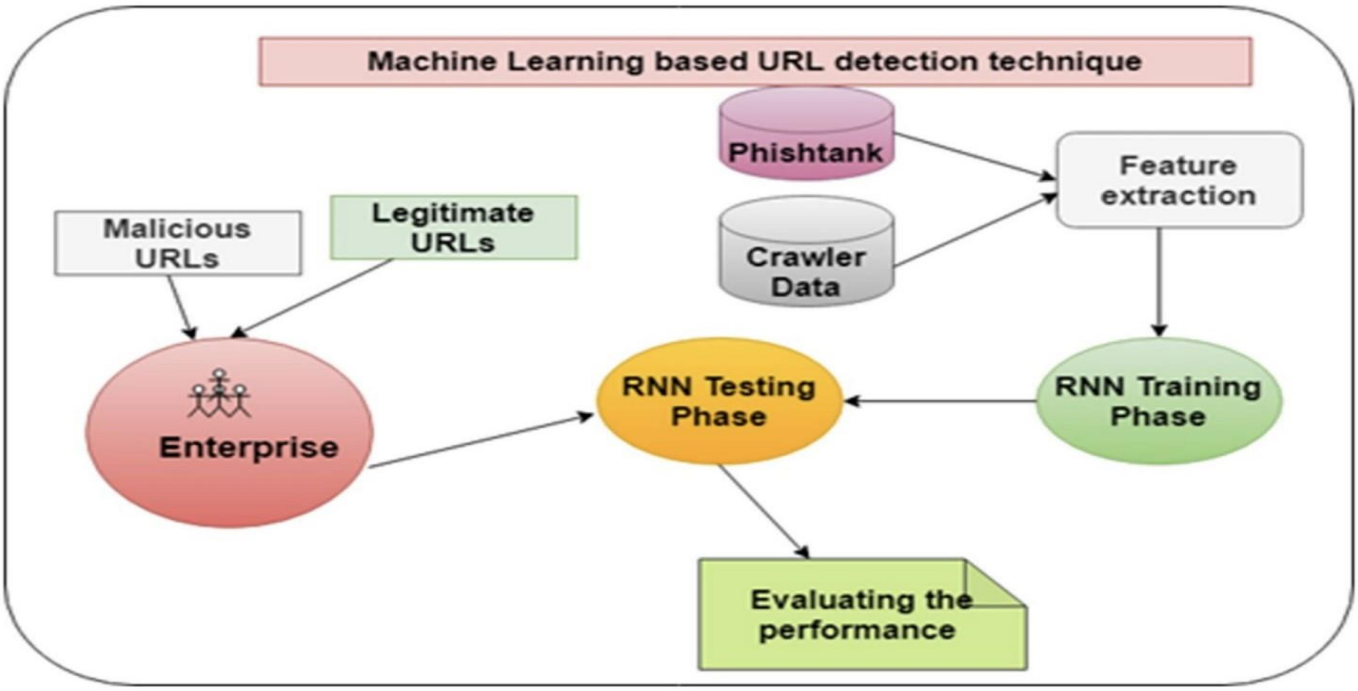
PROPOSED WORK

The term "recurring neural network" implies two broad groups of networks of a similar general structure, where one is a finite, and the other is an infinite input. Both network groups contain time dynamic behavior. A recurrent network of finite input is a directed acyclic graph that can be replaced by a purely feedforward neural network, whereas a recurrent network of infinite input is a directed cyclic graph that cannot be modified.

The modified versionof RNN is LSTM. It is a deep learning method, which prevents the gradient problem of RNN. Multiple gates are employed for improving the performance of LSTM. In comparison with RNN, LSTM prevents back propagation.
Each input of LSTM generates an output that becomes an input for the following layer or module of LSTM.

During the training phase, RNN stores the properties Pm and Pl to learn the environment. Moreover, each URL of the dataset from Phish tank and crawled URL is utilized in a way to instruct the model. Algorithms presents the stepsinvolved in the data collection and pre-process, correspondingly. The training phase uses the labels to train RNN to learn the malicious and legitimate URLs. Thus, the testing phase of the proposed RNN model receives each URL andpredicts the type of URL. RNN (LSTM) is developed with Python 3.0 in Windows 10 environment with the support of i7 processor.

LSTM model is an effective predictive model. It generates an output based on the arbitrary number of steps. There are five essential components that enables the model to produce long—term and short—term data



There are five essential components that enables the model to producelong—term and short—term data:

Cell state (CS)—It indicates the cell space that accommodate both longterm and short-term memories.

Hidden state (HS)—This is the output status information that user use todetermine URL with respect to the current data, hidden condition and current cell input.

Secret state (SS) -It is used to recover both short-term and long-termmemory, in order to make a prediction.

$$\sum (M + L) = x_n$$

$$\text{Input} = \sum_{n=0}^{m} x_n$$

$$\text{Malicious} = \text{Output\_RNN}(\text{Input}(\text{Pm}))$$

$$\text{Legitimate} = \text{Output\_RNN}(\text{Input}(\text{Pl}))$$

Input gate (IT) -The total number of information flows to the cell state.

Forget fate (FT) – The total number of data flows from the current input and past cell state to the present cell state.

Output gate (OT) -The total number of information flows to the hidden.

Explanation:

Input Gate- It identifies an input value for memory alteration. Sigmoid defines the values that can be up to 0,1. And the tanh function weights thevalues passed by, evaluating their significance from-1 to 1. Eqs 5 and 6 represents the input gate and cell state, respectively. Wn is the weight, HTt−1 is the previous state of hidden state, xt is the input, and bn is the bias vector which need to be learnt during the training phase. CT is calculated using tanh function.

$$IT = \partial(W_n(HT_{t-1}, x_t) + b_n)$$

$$CT = \tanh(W_d(HT_{t-1}, x_t) + b_c)$$

Forget gate- It finds out the necessary block information to be discarded from the memory. The sigmoid function is used to describe it. Eq 7contains (HTt−1) and content(xt) are examined, and the number of outputs between 0 and 1 is verified by each cell state CTt−1 number

$$FT = \partial\left(W_f(HT_{t-1}, x_t) + b_f\right)$$

Output gate- The input and the memory of the block is used to determine the output. Sigmoid function determines which values to let through 0 and 1. The tanh function presents weightage to the values which are transferred to determine their degree of importance rangingfrom-1 to 1 and multiplied with output of Sigmoid.

$$OT = \partial(W_o(HT_{t-1}, x_t) + b_o)$$

$$HT = O_t * \tanh(C_t)$$

Using some benchmark dataset, the accuracy of phishing detection systems is usually evaluated. The familiar phishing dataset totrain the ML based techniques are as follows:

Normal dataset: Alexa Rank is used as a benign and natural website benchmarking dataset. It is a commercial enterprise which carries out web data analysis. It obtains the browsing habits of users from different sources and analyses them objectively for thereporting and classification of Internet web-based URLs. Researchers use the rankings provided by Alexa to collect a number of high standard websites as the normal dataset to test and classify websites. presents the dataset in the form of a raw text file whereeach line in the order ascended mentions the grade and domain name of a website.

Phishing dataset: Phish tank is a familiar phishing website benchmark dataset which is available at https://phishtank.org/. It is agroup framework that tracks websites for phishing sites. Various users and third parties send alleged phishing sites that are ultimately selected as legitimate site by a number of users. Thus, phish tank offers a phishing website dataset in real time.

Researchers to establish data collection for testing and detection of Phishing websites use Phish tank's website. Phish tank dataset is available in the Comma Separated Value (CSV) format, with descriptions of a specific phrase used in every line of the file. Thesite provides details include ID, URL, time of submission, checked status, online status and target URLs

Phish tank and Crawler are used to collect Malicious and Benign URLs. A crawler is developed in order to collect URLs from Alexa Rank website. Alexa Rank publishes set of URLs with ranking to support to research community. The crawler crawled a number of 7658 URLs from Alexa Rank between June to November 2020. 6042 URLs were collected through Phish tank datasets.

Data Collection- This algorithm represents the processes involved in data collection. Data Repositories such as Phish tank and Crawler are used to collect Malicious and Benign URLs. A crawler is developed in order to collect URLs from Alexa Rank website. Alexa Rank publishes set of URLswith ranking to support to research community. In this study, the crawler crawled a number of 7658 URLs from Alexa Rank between June 2020 to November 2020. 6042 URLs were collected through Phish tank datasets. During the data collection, extracted data are stored in W and returned as W1 with number of URLS.

Input: Data Repositories
Output: Raw Data

1: **procedure** DATA COLLECTION
2: $W \leftarrow$ ExtractData(Repositories)
3: $W1 \leftarrow$ FilterInvalidURL(W)
4: $N \leftarrow$ Count(W1)
5: **return** $W1$,N
6: **end procedure**

Data Pre-process- This algorithm illustrates the steps of data pre—process. url is one of the elements of URL dataset. In this process, the raw data is pre—processed by scanning each URL in the dataset. A set of functions are developed in order to remove the irrelevant data. Finally, D2 is the set of features returned by the pre—process activity.

Input: Raw Data
Output: Features

1: **procedure** DATA PRE-PROCESS
2: **while** $url \leftarrow URL$ **do**
3: $D \leftarrow$ RemoveProtocols(RawData)
4: $D1 \leftarrow$ RemoveIrrelevant(D)
5: $D2 \leftarrow$ RemoveDomainName(D1)
6: **end while**
7: **return** $D2$
8: **end procedure**

Data Transformation- This algorithm represents the processes of data transformation. “Num” is the vector returned by the data transformation process. During this process, each feature of D2 is converted as a vector. Each data in D2 is processed using the Generate Vectors function. A vector is generated and passed as an input to the training phase.

```
Input: Features(D2)
Output: Vectors
procedure DATA TRANSFORMATION(D2)
    while d ← D2 do
        Num ← GenerateVectors(d)
    end while
    return Num
end procedure
```

Training Phase- In the training phase. Each URL is processed with the support of vector. LSTM Lib is one of the functions in the LSTM to predict an output using the vectors. The library is updated with the extracted features that contains the necessary data related to malicious and normal web pages. Thus, the iterative process is used to scan each vector and suspicious URL and generate a final outcome. Lastly, op is the prediction returned by the proposed method during the training phase.

```
Input: Vector(Num), URL
Output: URL-Type
procedure TRAINING PHASE(Num)
    while num ← Num do
        if num = Feature(URL) then
            op = Phishing URL
        else
            op = Legitimate URL
            if op = LSTMLib(feature) then
                op = Phishing URL
            else
                op = Legitimate URL
            end if
        end if
    end while
    return op
end procedure
```

Testing Phase- The algorithm indicates the testing phase of the proposed URL detection. The proposed processes each element from LSTM Memory function is compared with the vector of URL and decide an output. The f is the element of the feedback which is collected from the crawler that indicates the page rank of a website. The page rank indicates the value of a website and the lowest ranking website will be declared as malicious or suspicious to alert the users.

Input: URL
Output: Type of URL

```
1: procedure TESTING PHASE(URL)
2:     while url ← URL do
3:         if element ← LSTMMemory = Feature(URL) then
4:             op = Phishing URL
5:         else
6:             op = Legitimate URL
7:             feedback = phishtank(op)
8:             if element ← LSTMMemory = f ← feedback then
9:                 op = Phishing URL
10:            else
11:                op = Legitimate URL
12:            end if
13:        end if
14:    end while
15:    return op
16: end procedure
```

Epoch settings- The epoch settings are the part of the training phase. The epoch value is used to indicate the execution time of a method. The learning rate can be increased to improve the performance of a method.

```
for epoch in total_range(no_of_epochs):
  new_decay = orig_decay ** max(epoch + 1 - max__epoch, 0.0)
  m.a_lr(sess, learning_rate * new_lr_decay)
  current_state = np.zeros((num_layers, 2, batch_size, m.hidden_size))
  for step in range(training_input.epoch_size):
    if step % 50 != 0:
      cost, _, current_state = sess.run([m.cost, m.train_op, m.state],
                                        feed_dict={m.init_state: current_state})
    else:
      cost, _, current_state, acc = sess.run([m.cost, m.train_op, m.state, m.accuracy],
                                             feed_dict={m.init_state: current_state})
      print("Epoch {}, Step {}, cost: {:.3f}, accuracy: {:.3f}".format(epoch, step, cost, acc))
```

PRACTICAL IMPLEMENTATION

```
[ ]  import pandas as pd
     import numpy as np
     from collections import defaultdict
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from tensorflow.keras.preprocessing import sequence
     from prettytable import PrettyTable
     import tensorflow as tf
```

```
     # Some high level parameters:
     show_top_n = 20
     random_seed = 16

     # Setting the random seed so that the code is repeatable.
     np.random.seed(random_seed)
```

+ Code

```
     # Loading the current combined online-valid dataset.
     # Use the dataset_downloader.py to download the current Phishtank online-valid.csv and update the combined dataset.
     print("Phishtank Online Valid Dataset")
     online_valid_df = pd.read_csv("combined_online_valid.csv")
     online_valid_df
```

```
[ ]  # Extracting tld and domain name.
     tld_count = defaultdict(lambda: 0)
     domain_names = []
     for index, row in online_valid_df.iterrows():
         # Extracting the tld from the url
         domain_name = row["url"].replace("https://","").replace("http://","").split("/")[0]
         domain_names.append(domain_name)
         tld = domain_name.split(".")[-1]
         tld_count[tld] += 1

     tld_df = pd.Series(dict(tld_count))
     tld_df.sort_values(ascending=False, inplace=True)
     tld_print = tld_df.iloc[:show_top_n]
     tld_print["OTHERS"] = tld_df.iloc[show_top_n:].sum()
```

```
[ ]  # Adding the domain names extracted from the phishing urls as a new column.
     online_valid_df["domain_names"] = domain_names
```

```
[ ]  # Loading the whitelist from the 1 million most frequently visited domains.
     whitelist_file_umbrella = "top-1m_umbrella.csv"
     whitelist_df = pd.read_csv(whitelist_file_umbrella, header=None, names=["rank", "domain_names"])
```

```
[ ]  # Finding if there are any domains that are also in the whitelist.
     domains_in_whitelist = np.intersect1d(online_valid_df["domain_names"], whitelist_df["domain_names"])
     # Tagging the whitelisted domains as such.
     online_valid_df["in_whitelist"] = np.in1d(online_valid_df["domain_names"], domains_in_whitelist)
```

```
[ ]  # Printing some data examples for reference.
     print(online_valid_df.shape[0], "rows")
     print(tld_print.to_frame(name="TLD Count").transpose().to_string())
     print(f"Percentage of top {show_top_n} tlds: {np.round(100*tld_df.iloc[:show_top_n].sum()/tld_df.sum(), decimals=2)} %")
     online_valid_df.head(20)
```

```
# Printing the top of the whitelist.
print("Whitelist file:", whitelist_file_umbrella)
print(whitelist_df.shape[0], "rows")
print(whitelist_df.head(20))
```

```
Whitelist file: top-1m_umbrella.csv
1000000 rows
    rank                     domain_names
0      1                       google.com
1      2                   www.google.com
2      3                    microsoft.com
3      4                    facebook.com
4      5                     netflix.com
5      6                 windowsupdate.com
6      7                   ftl.netflix.com
7      8               prod.ftl.netflix.com
8      9                 data.microsoft.com
9     10             api-global.netflix.com
10    11            ctldl.windowsupdate.com
11    12          nrdp.prod.ftl.netflix.com
12    13                   doubleclick.net
13    14                 g.doubleclick.net
14    15       safebrowsing.googleapis.com
15    16  settings-win.data.microsoft.com
16    17                       youtube.com
17    18       googleads.g.doubleclick.net
18    19         events.data.microsoft.com
19    20                          live.com
```

```
[ ]  # How many domains are in both the whitelist and in the phishing urls?
     print()
     print("Number of urls that have domains which are in the whilelist:", online_valid_df["in_whitelist"].sum())
```

```
Number of urls that have domains which are in the whilelist: 1442
```

```
# Encoding code/idea from TensorFlow 2.0 Complete Course - Python Neural Networks for Beginners Tutorial freeCodeCamp.org
# https://colab.research.google.com/drive/1ysEKrw_LE2jMndo1snrZUh5w87LQsCxk#forceEdit=true&sandboxMode=true
vocab = sorted(set("".join(X)), reverse=True)
# Inserting a space at index 0, since it is not used in url and will be used for padding the examples.
vocab.insert(0, " ")
vocab_size = len(vocab)

print()
print(f"Encoding Vocabulary ({vocab_size}) used:")
print(vocab)
# Creating a mapping from unique characters to indices
char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)

def text_to_int(text):
  return np.array([char2idx[c] for c in text])

print("Encoding example:")
print(text_to_int(phishing_domains[0]))

def int_to_text(ints):
  try:
    ints = ints.numpy()
  except:
    pass
  return ''.join(idx2char[ints])

print(int_to_text(text_to_int(phishing_domains[0])))
```

```
Encoding Vocabulary (73) used:
[' ', 'z', 'y', 'x', 'w', 'v', 'u', 't', 's', 'r', 'q', 'p', 'o', 'n', 'm', 'l', 'k', 'j', 'i', 'h', 'g', 'f', 'e', 'd',
Encoding example:
[13 22  4 11  9 12 23  6 24  7  8  8  7 12  9 22 69 24 12 14]
newproductsstore.com
```

```
[ ] # Reducing how many samples to print so printouts dont get so big.
    show_top_n = 5
    print(f"Training and testing data: (showing first {show_top_n})")
    print(f"Train data {len(X_train)} samples")
    print(list(zip(X_train[:show_top_n], y_train[:show_top_n], sample_weights_train[:show_top_n])))
    print(f"Test data {len(X_test)} samples")
    print(list(zip(X_test[:show_top_n], y_test[:show_top_n], sample_weights_test[:show_top_n])))
```

```
Training and testing data: (showing first 5)
Train data 47374 samples
[('p.adnxs.com', 0, 0.6666666666666666), ('danar.net.pl', 0, 0.6666666666666666), ('tenzinngodup.com', 1, 1.0), ('metrolagu.ltd',
Test data 8361 samples
[('tcprelay.ztehome.com.cn', 0, 0.6666666666666666), ('televoip.com.ve', 1, 1.0), ('ransecuritysoc.com', 0, 0.6666666666666666), (
```

```
[ ] # Encoding the domain names using the vocabulary
    X_train_encoded = [text_to_int(domain_name) for domain_name in X_train]
    X_test_encoded = [text_to_int(domain_name) for domain_name in X_test]
    print()
    print(f"Encoded data: (showing first {show_top_n})")
    print(f"Train data {len(X_train_encoded)} samples, encoded")
    print(list(zip(X_train_encoded[:show_top_n], y_train[:show_top_n])))
    print(f"Test data {len(X_test_encoded)} samples, encoded")
    print(list(zip(X_test_encoded[:show_top_n], y_test[:show_top_n])))

    # Padding to the right sequence length.
    X_train_encoded_padded = sequence.pad_sequences(X_train_encoded, max_seq_len)
    X_test_encoded_padded = sequence.pad_sequences(X_test_encoded, max_seq_len)
    print()
    print(f"Encoded and padded data: (showing first {show_top_n})")
    print(f"Train data {len(X_train_encoded_padded)} samples, encoded")
    print(list(zip(X_train_encoded_padded[:show_top_n], y_train[:show_top_n])))
    print(f"Test data {len(X_test_encoded_padded)} samples, encoded")
    print(list(zip(X_test_encoded_padded[:show_top_n], y_test[:show_top_n])))
```

```
[ ] # https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM
    model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size, 64),
        tf.keras.layers.LSTM(128),
        tf.keras.layers.Dense(128,activation="tanh"),
        tf.keras.layers.Dense(1, activation="sigmoid")
    ])
```

```
    # Compiling the model
    model.compile(loss="binary_crossentropy", optimizer="adam", metrics=['acc'])
    print(model.summary())
```

```
[ ] class_weight={0: (1/(oversampling_rate+1)), 1: (oversampling_rate/(oversampling_rate+1))}
    print("Using the class weighting:", class_weight)
    # Training the model
    # Setting up callback to monitor the selected loss, and stops training if it doesnt improve for patience-number of epochs.
    # After stopping training will restore the weights from the best iteration on this value encountered so far.
    early_stopping_callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=4, restore_best_weights=True)
```

```
Using the class weighting: {0: 0.4, 1: 0.6}
```

```
[ ] history = model.fit(X_train_encoded_padded, y_train,
                        epochs=100,
                        validation_data=(X_test_encoded_padded, y_test),
                        class_weight=class_weight,
                        sample_weight=sample_weights_train,
                        callbacks=[early_stopping_callback])
```

```
Epoch 1/100
1481/1481 [==============================] - 10s 6ms/step - loss: 0.1902 - acc: 0.7214 - val_loss: 0.4513 - val_acc: 0.7696
Epoch 2/100
```

```
Phishing ULR examples:
Prediction on url: frgcxtmjawefgrthdcusge.dab 0.029374275
Prediction on url: evilmadeupurl.phish 0.6535125
Prediction on url: evil.madeupurl.phish 0.18197966

Safe URL examples:
Prediction on url: google.com 0.04181947
Prediction on url: www.google.com 0.09977573
Prediction on url: gmail.google.com 0.0022916235
Prediction on url: mail.google.com 0.0011315699
Prediction on url: tudelft.nl 0.07059877
Prediction on url: brightspace.tudelft.nl 0.9962966
Prediction on url: colab.research.google.com 0.0012517852
Prediction on url: 00-gayrettepe-t3-8---00-gayrettepe-xrs-t2-1.statik.turktelekom.com.tr 0.16847374
```

```
# Custom evaluate
best_threshold = threshold_evaluation_plotter(X_test_encoded_padded, y_test)
mean_prediction = evaluate_nn_model(X_test_encoded_padded, y_test, threshold=best_threshold)
```

RESULTS AND SNAPSHOTS

```
Best performance at threshold: 0.6967336683417086
Calculated 8361 predictions with a mean value of 0.4498916268348694
Evaluating using threshold 0.6967336683417086
Cut-off threshold: 0.6967
Evaluation counts: {'TN': 4517, 'FP': 525, 'FN': 544, 'TP': 2775}
+------------------+----------------+--------------------+
| Accuracy 87.214% | Predicted safe | Predicted phishing |
+------------------+----------------+--------------------+
|   Not phishing   |    TN: 4517    |      FP: 525       |
|                  |  NPV: 89.251%  |    FDR: 15.909%    |
|                  |  TNR: 89.587%  |    FPR: 10.413%    |
|   Is phishing    |     FN: 544    |      TP: 2775      |
|                  |  FOR: 10.749%  |    PPV: 84.091%    |
|                  |  FNR: 16.39%   |    TPR: 83.61%     |
+------------------+----------------+--------------------+
```

Examples for TN Bin range: 0.00064611493 - 0.13979572 , Num. Samples: 3428

	input	ground truth	prediction
0	eoelijh.tj	0	0.005883
1	m39.czhilni.com	0	0.001349
2	m15.rxzsodk.com	0	0.000750
3	a3.public.asfom-b-2.prod.infra.webex.com	0	0.001815
4	service.gmx.net	0	0.002599
5	scmewdfw2wdfw2-357-wdfw2-public.wbx2.com	0	0.005711
6	r6---sn-gxuog0-axqe.googlevideo.com	0	0.001071
7	guru.sgslb.sanook.com	0	0.011387
8	android.api.itools.cn	0	0.000809
9	nedigital.sg	0	0.021610
10	students-residents.aamc.org	0	0.003656
11	server-43.bczg.ru	0	0.001131
12	changan.com.cn	0	0.010027
13	play.googleapis.com.myanmarnet.com	0	0.001417
14	img3m1.ddimg.cn	0	0.002048

Examples for TN Bin range: 0.13979572 - 0.27894533 , Num. Samples: 330

	input	ground truth	prediction
0	springernature.io	0	0.264976
1	mail.frenckengroup.com	0	0.176848
2	telematicaitalia.it	0	0.166007
3	altamoreefontani.it	0	0.243631
4	root.secureserver.net	0	0.256539
5	dc.applicationinsights.azure.us	0	0.209147
6	elements-downloads.envatousercontent.com	0	0.246847
7	js-sdk.analytics-data.io	0	0.149199
8	1utama.com.my	0	0.144151
9	przedszkole.staporkow.edu.pl	0	0.214979
10	mediaaws.almasryalyoum.com	0	0.193101
11	www.pbt.com	0	0.189702
12	jh.cz	0	0.176616
13	mutmx01pxi.proxxi.com.br	0	0.194653
14	newcrest.com	0	0.249903

Examples for TN Bin range: 0.41809493 - 0.55724454 , Num. Samples: 256

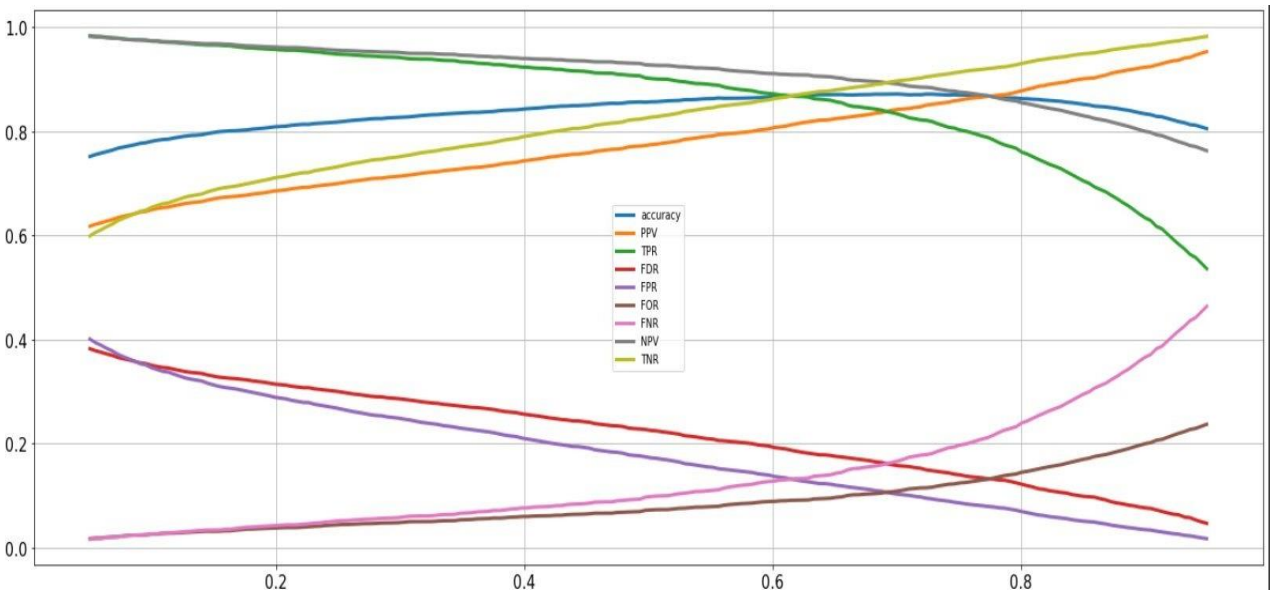
	input	ground truth	prediction
0	salvationarmy.org	0	0.499633
1	g-session.monitor.core.app.alexa.a2z.com	0	0.453978
2	www.hbogoasia.id	0	0.450224
3	iaasep.info	0	0.488283
4	vodplhq.cdn021playhq.xyz	0	0.531188
5	vkamfdpifst.info	0	0.528796
6	f2pool.info	0	0.449684
7	eagle.ptialaska.net	0	0.509918
8	www.zeiss.com	0	0.448118
9	zipcar.io	0	0.436801
10	xjapanese.com	0	0.424095
11	tigerto.com	0	0.528126
12	yphimsex.cc	0	0.432384
13	dynarechi.com	0	0.445684
14	agatgroup.com	0	0.428187

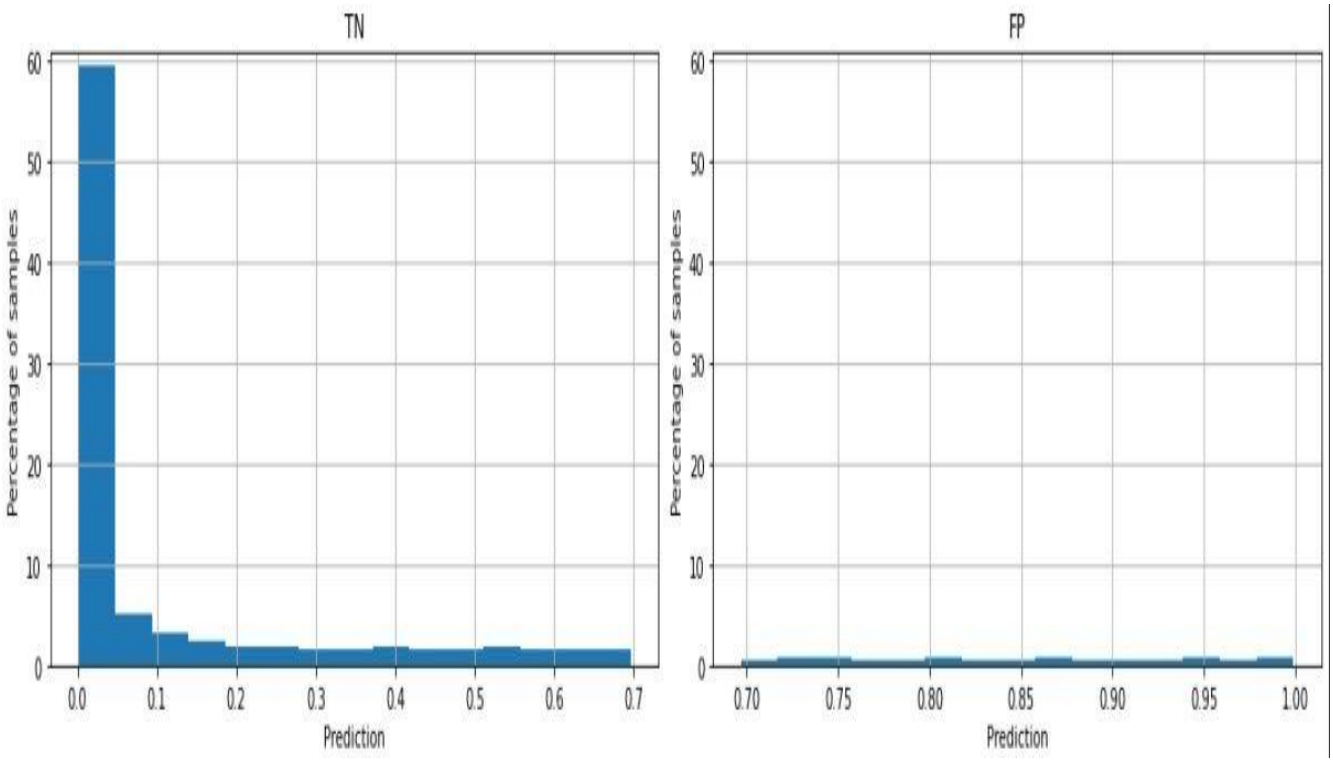
Examples for FP Bin range: 0.87813526 - 0.93859726 , Num. Samples: 102

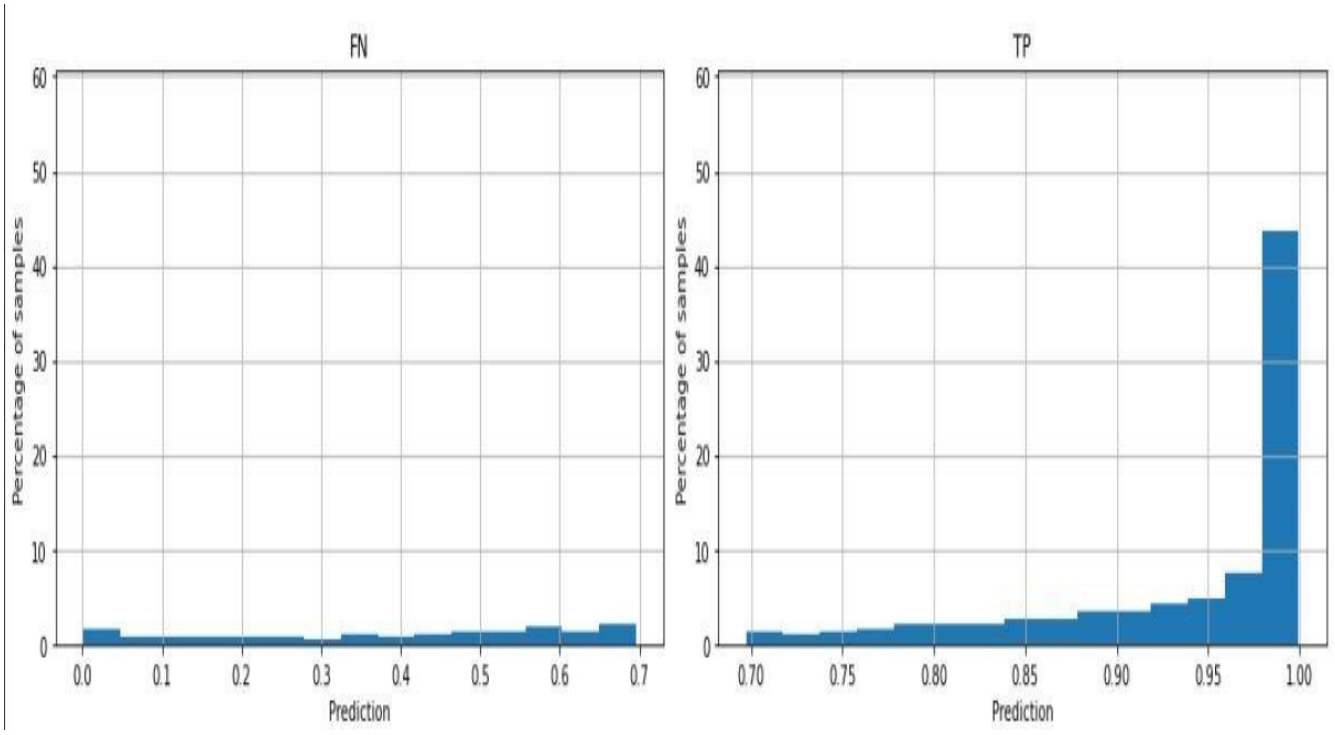
	input	ground truth	prediction
0	www.seamanjobsite.com	0	0.880125
1	goldenbellsdelhi.com	0	0.920386
2	ldsbkkw001.ldschurch.org	0	0.878623
3	www.amalgamatedbank.com	0	0.892042
4	www.mobil123.com	0	0.912669
5	daemon-tools.cc	0	0.904496
6	currykryss.se	0	0.897703
7	ios-analytics-prodhome1.mysoluto.com	0	0.926272
8	americashloan.club	0	0.915034
9	itrack.top	0	0.919335
10	tr.slvrbullet.com	0	0.906971
11	sparkpostelite.com	0	0.931842
12	www.clintonfoundation.org	0	0.896093
13	dakota.net	0	0.908519
14	brainly.in	0	0.912683

Examples for TP Bin range: 0.9389703 - 0.999358 , Num. Samples: 1858

	input	ground truth	prediction
0	infomailer.temp.swtest.ru	1	0.991543
1	z-dot-cedar-code-289917.nn.r.appspot.com	1	0.999229
2	umconnectumt-my.sharepoint.com	1	0.943072
3	rakutem-naladuna.cc	1	0.990952
4	identiity-sociiety-inc.ga	1	0.994718
5	-dot-cryptic-now-290917.ey.r.appspot.com	1	0.999161
6	51jianli.cn	1	0.989384
7	www.smbcc-card.top	1	0.998046
8	ee-billingservice.com	1	0.966228
9	andreacostafisio.com.br	1	0.986412
10	www.stolizaparketa.ru	1	0.996789
11	mazeadvokater-my.sharepoint.com	1	0.954561
12	email.account.com.fqjnrf.cn	1	0.996187
13	unitus.mk.ua	1	0.970483
14	expeditions-of-e.com	1	0.957091




TN
FP
Percentage of samples
Prediction
0
10
20
30
40
50
60
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.70
0.75
0.80
0.85
0.90
0.95
1.00


FN
TP
Percentage of samples
Prediction
0
10
20
30
40
50
60
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.70
0.75
0.80
0.85
0.90
0.95
1.00

APP.PY:

app.py 8 training.py 7

D: > Desktop > app.py > ...

```python
import streamlit as st
from keras.models import load_model
import numpy as np
import os
from streamlit_lottie import st_lottie

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import pandas as pd
import numpy as np
import json
import tensorflow as tf
from keras.preprocessing import sequence

class phishing_ai():
    """
    Wrapper class to add custom prediction function to the model.
    """

    def __init__(self, model_name='phishing_detection_model'):
        with open(f'{model_name}_settings.json', "r") as msf:
            self.model_settings = json.load(msf)
        self.char2idx = {u:i for i, u in enumerate(self.model_settings["vocab"])}
        self.model = tf.keras.models.load_model(model_name)

    def text_to_int(self, text):
        return np.array([self.char2idx[c] for c in text])

    def is_phishing(self, url):
        """
        Returns True if phishing, False if not-phishing
        """
        encoded_text = sequence.pad_sequences([self.text_to_int(url)], self.model_settings["max_seq_len"])
        result = self.model.predict(encoded_text)
        return result[0][0] > self.model_settings["threshold"]
```

app.py 8 training.py 7

D: > Desktop > app.py > ...

```python
st.set_page_config(page_title="Phishing Website Detector", page_icon=":tada:", layout="wide")

def local_css(file_name):
    with open(file_name) as f:
        st.markdown(f"<style>{f.read()}</style>", unsafe_allow_html=True)
local_css("style/style.css")

st.title("Welcome to Phishing Website Detection")
st.write(
        "TRUE: Given URL is Phishing"
        )
st.write(
        "FALSE: Given URL is not Phishing"
    )

url = st.text_input("Enter the URL to Check")

btn = st.button("Check Safety")

if btn:
    #st.text("Safe")
    tf.config.set_visible_devices([], 'GPU')
    phishing_ai = phishing_ai()
    print("Prediction on url:", url, phishing_ai.is_phishing(url))
    st.subheader(phishing_ai.is_phishing(url))

with st.container():
    st.write("---")
    left_column, right_column = st.columns(2)
    with left_column:
        st.header("PROJECT DETAILS")
        #st.write("##")
        st.write(
            """
            A PROJECT ON PHISHING WEBSITE DETECTION UNDER THE GUIDANCE OF:
            - PROF. CHANDRA MOHAN B
            TEAM MEMBERS:
            - KUSHAJ ARORA 20BCI0047
            - SHASHWAT KUMAR 20BCE2818
            - ANANNYA CHULI 20BCE2046
            """
        )
```

TRAINING.PY:

app.py 8 training.py 7

D: > Desktop > training.py > ...

```python
import pandas as pd
import numpy as np
from collections import defaultdict
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.preprocessing import sequence
from prettytable import PrettyTable
import tensorflow as tf
import json

# Some high level parameters:
show_top_n = 20
random_seed = 16
max_seq_len = 40
model_name='phishing_detection_model'
# Set oversampling_rate to 1 to have the positive samples match the phishing samples. Set to greater than 1 to use more negative samples.
oversampling_rate = 1.25

# Setting the random seed so that the code is repeatable.
np.random.seed(random_seed)

# Loading the current combined online-valid dataset.
# Use the dataset_downloader.py to download the current Phishtank online-valid.csv and update the combined dataset.
print("Phishtank Online Valid Dataset")
online_valid_df = pd.read_csv("combined_online_valid.csv")

# Extracting tld and domain name.
tld_count = defaultdict(lambda: 0)
domain_names = []
for index, row in online_valid_df.iterrows():
    # Extracting the tld from the url
    domain_name = row["url"].replace("https://","").replace("http://","").split("/")[0]
    domain_names.append(domain_name)
    tld = domain_name.split(".")[-1]
    tld_count[tld] += 1

tld_df = pd.Series(dict(tld_count))
tld_df.sort_values(ascending=False, inplace=True)
tld_print = tld_df.iloc[:show_top_n]
tld_print["OTHERS"] = tld_df.iloc[show_top_n:].sum()
```

app.py 8 training.py 7

D: > Desktop > training.py > ...

```python
# Tagging the whitelisted domains as such.
online_valid_df["in_whitelist"] = np.in1d(online_valid_df["domain_names"], domains_in_whitelist)

# Printing some data examples for reference.
print(online_valid_df.shape[0], "rows")
print(online_valid_df.head(20))
print(tld_print.to_frame(name="TLD Count").transpose().to_string())
print(f"Percentage of top {show_top_n} tlds: {np.round(100*tld_df.iloc[:show_top_n].sum()/tld_df.sum(), decimals=2)} %")

# Printing the top of the whitelist.
print()
print("Whitelist file:", whitelist_file_umbrella)
print(whitelist_df.shape[0], "rows")
print(whitelist_df.head(20))

# How many domains are in both the whitelist and in the phishing urls?
print()
print("Number of urls that have domains which are in the whilelist:", online_valid_df["in_whitelist"].sum())

# For the dataset, excluding all where the domain name is in the whitelist.
# Since if the domain is in both lists we cannot tell if its safe or phishing? --> This helps with labeleling the data.
online_valid_df_without_intersection = online_valid_df.loc[online_valid_df['in_whitelist'] == False]
whitelist_df_without_intersection = whitelist_df.loc[np.invert(whitelist_df['domain_names'].isin(domains_in_whitelist))]

# Getting the array of all phishing domain names.
phishing_domains = online_valid_df_without_intersection["domain_names"].values
# Randomly sample a number of safe urls, sice the ratio of classes in the training data should not be too much out of balance.
whitelist_domains = np.random.choice(whitelist_df_without_intersection["domain_names"].values, size=int(oversampling_rate*len(phishing_domains)), replace=False)

print()
print("Selected Data Examples:")
print("Phishing domains:", phishing_domains, len(phishing_domains))
print("Benign domains:", whitelist_domains, len(whitelist_domains))

# Calling a phishing url 1 and a not-phishing url 0.
# Using character encoding as the vocabulary.
# Feeding the url as the sequence.
# Creating the samples array and the label array
print()
X = list(phishing_domains) + list(whitelist_domains)
y = [1]*len(phishing_domains) + [0]*len(whitelist_domains)
sample_weights = [1]*len(phishing_domains) + [1/oversampling_rate]*len(whitelist_domains)
```

app.py 8 training.py 7

D: > Desktop > training.py > ...

```python
    # Creating the samples array and the label array
    print()
    X = list(phishing_domains) + list(whitelist_domains)
    y = [1]*len(phishing_domains) + [0]*len(whitelist_domains)
    sample_weights = [1]*len(phishing_domains) + [1/oversampling_rate]*len(whitelist_domains)


    vocab = sorted(set("".join(X)), reverse=True)
    # Inserting a space at index 0, since it is not used in url and will be used for padding the examples.
    vocab.insert(0, " ")
    vocab = np.array(vocab)
    vocab_size = len(vocab)

    print()
    print(f"Encoding Vocabulary ({vocab_size}) used:")
    print(vocab)
    # Creating a mapping from unique characters to indices
    char2idx = {u:i for i, u in enumerate(vocab)}
    idx2char = vocab

    def text_to_int(text):
      return np.array([char2idx[c] for c in text])

    print("Encoding example:")
    print(text_to_int(phishing_domains[0]))

    def int_to_text(ints):
      try:
        ints = ints.numpy()
      except:
        pass
      return ''.join(idx2char[ints])

    print(int_to_text(text_to_int(phishing_domains[0])))


    # Investigating the domain name length for the combined domain names:
    X_elem_len = [len(domain_name) for domain_name in X]
    print(sorted(X_elem_len, reverse=True)[:show_top_n])

    # Setting some max length for our urls.
    print((np.array(X_elem_len) > max_seq_len).sum(), "URLs longer than the cutoff length", max_seq_len)
```

app.py 8 training.py 7

D: > Desktop > training.py > ...

```python
    # Investigating the domain name length for the combined domain names:
    X_elem_len = [len(domain_name) for domain_name in X]
    print(sorted(X_elem_len, reverse=True)[:show_top_n])

    # Setting some max length for our urls.
    print((np.array(X_elem_len) > max_seq_len).sum(), "URLs longer than the cutoff length", max_seq_len)

    # Creating test and training datasets
    print()

    X_train, X_test, y_train, y_test, sample_weights_train, sample_weights_test = train_test_split(np.array(X),
                                                                                                   np.array(y),
                                                                                                   np.array(sample_weights),
                                                                                                   test_size=0.15,
                                                                                                   random_state=random_seed)


    # Reducing how many samples to print so printouts dont get so big.
    show_top_n = 5
    print(f"Training and testing data: (showing first {show_top_n})")
    print(f"Train data {len(X_train)} samples")
    print(list(zip(X_train[:show_top_n], y_train[:show_top_n], sample_weights_train[:show_top_n])))
    print(f"Test data {len(X_test)} samples")
    print(list(zip(X_test[:show_top_n], y_test[:show_top_n], sample_weights_test[:show_top_n])))

    # Encoding the domain names using the vocabulary
    X_train_encoded = [text_to_int(domain_name) for domain_name in X_train]
    X_test_encoded = [text_to_int(domain_name) for domain_name in X_test]
    print()
    print(f"Encoded data: (showing first {show_top_n})")
    print(f"Train data {len(X_train_encoded)} samples, encoded")
    print(list(zip(X_train_encoded[:show_top_n], y_train[:show_top_n])))
    print(f"Test data {len(X_test_encoded)} samples, encoded")
    print(list(zip(X_test_encoded[:show_top_n], y_test[:show_top_n])))

    # Padding to the right sequence length.
    X_train_encoded_padded = sequence.pad_sequences(X_train_encoded, max_seq_len)
    X_test_encoded_padded = sequence.pad_sequences(X_test_encoded, max_seq_len)
    print()
    print(f"Encoded and padded data: (showing first {show_top_n})")
    print(f"Train data {len(X_train_encoded_padded)} samples, encoded")
    print(list(zip(X_train_encoded_padded[:show_top_n], y_train[:show_top_n])))
    print(f"Test data {len(X_test_encoded_padded)} samples, encoded")
    print(list(zip(X_test_encoded_padded[:show_top_n], y_test[:show_top_n])))
```

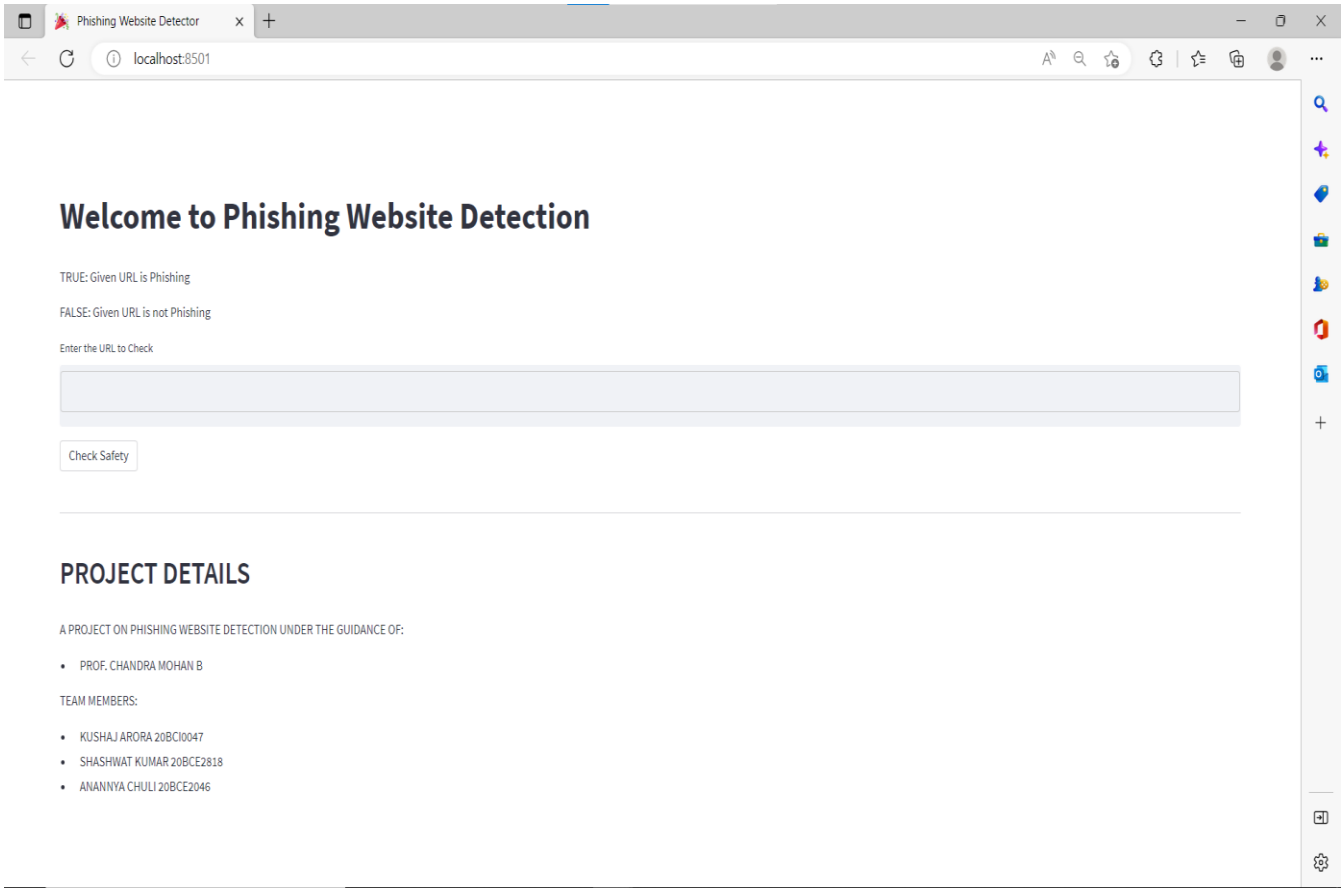
```python
# Creating the recurrent model for the predictions:
print("\n---------------Tensorflow magic-----------------\n")
#Evaluating the model
def evaluate_nn_model(X, y, threshold=0.5, bins=5, graph_bins=15, examples_per_bin=15):
    """
    Custom nn evaluation to get the TP, TN, FP, FN rates.
    Anything below threshold is considered not phishing.
    Anything above threshold is considered phishing.
    """
    predictions = model.predict(X).flatten()
    mean_prediction = np.mean(predictions)
    print(f"Calculated {len(predictions)} predictions with a mean value of {mean_prediction}")
    print(f"Evaluating using threshold {threshold}")
    # Turning the predictions into 0 and 1 by checking the threshold. (0 safe, 1 phishing)
    predictions_boolean = predictions > threshold
    predictions_binary = predictions_boolean.astype(np.int)
    print(f"Cut-off threshold: {np.round(threshold, decimals=4)}")
    groundtruth_elements, groundtruth_counts = np.unique(y, return_counts=True)
    groundtruth_counts = dict(zip(groundtruth_elements, groundtruth_counts))
    evaluation_ratios_counts, sample_outcomes = statistics_evaluator(predictions_binary, y)
    statistics_table_printer(evaluation_ratios_counts)
    # showing some examples for each type of outcome: 0 TN, 1 FP, 2 FN, 3 TP
    fig, axs = plt.subplots(2, 2, figsize=(15, 8))
    outcome_index = [0, 1, 2, 3]
    outcome_plot_positions = [0, 1, 2, 3]
    outcome_labels = ["TN", "FP", "FN", "TP"]
    y_axis_max = 0
    for outcome in outcome_index:
        outcome_indexes = np.where(np.array(sample_outcomes) == outcome)[0]
        # Instead of random samples, do a histogram with bins of the predictions for this outcome.
        # Then sample examples from each bin.
        outcome_predictions = predictions[outcome_indexes]
        outcome_binary = [ int(ind) for ind in list(str(bin(outcome_plot_positions[outcome])).replace("0b","").rjust(2, "0"))]
        outcome_hist, outcome_bins = np.histogram(outcome_predictions, bins=bins)
        plot_hist, plot_bins = np.histogram(outcome_predictions, bins=graph_bins)
        outcome_total_count = groundtruth_counts[outcome_binary[0]]
        plot_hist = (100*np.array(plot_hist))/outcome_total_count
        axs[outcome_binary[0], outcome_binary[1]].bar(plot_bins[:-1], plot_hist, width = plot_bins[1]-plot_bins[0], align="edge")
        y_axis_max = max( max(plot_hist), y_axis_max)
        axs[outcome_binary[0], outcome_binary[1]].set_title(outcome_labels[outcome])
        # Randomly sample some examples from each bin for this outcome:
        for bin_start, bin_end in zip(outcome_bins[:-1], outcome_bins[1:]):
```

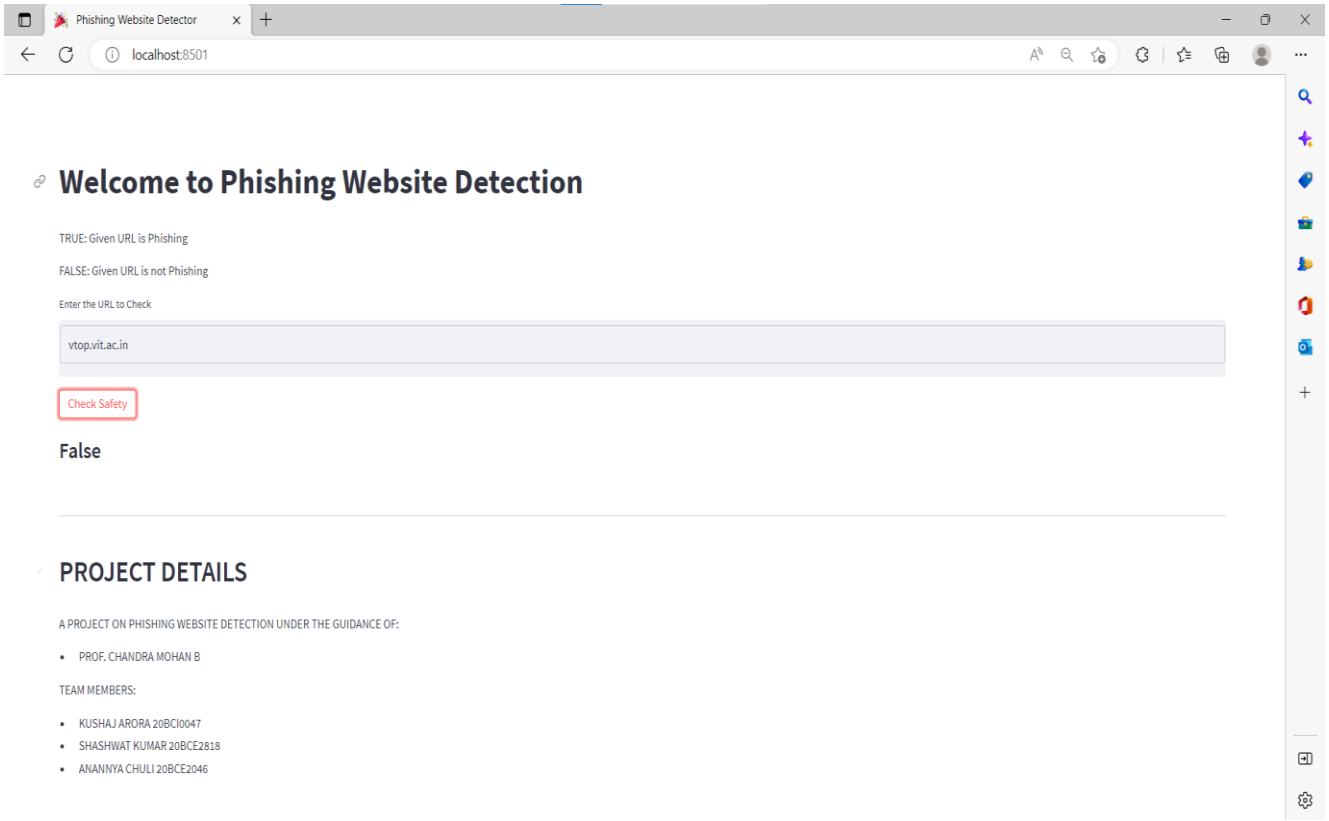
```python
def statistics_evaluator(predictions_binary, y_binary):
    # Concattenating the strings of the binary value of the prediction and the truth.
    # First value is the prediction, second the actual label
    # Hypothesis is: is phishing -> positive: yes phishing, negative: no phishing
    # Then 00 would be a TN, 01 is a FP, 10 is a FN, 11 is a TP.
    # Converting the binary outcomes to integer: 0 TN, 1 FP, 2 FN, 3 TP
    hypothesis_tests = [int(str(label)+str(prediction), 2) for prediction, label in zip(predictions_binary, y_binary)]
    # Counting the number of times each unique value in the tests is returned.
    unique_elements, counts_elements = np.unique(hypothesis_tests, return_counts=True)
    counts_elements = dict(zip(unique_elements, counts_elements))
    outcome_labels = ["TN", "FP", "FN", "TP"]
    evaluation_ratios_counts = dict(zip(outcome_labels, [counts_elements.get(0, 0), counts_elements.get(1, 0), counts_elements.get(2, 0), counts_elements.get(3, 0)]))
    return evaluation_ratios_counts, hypothesis_tests

def statistics_table_printer(evaluation_ratios_counts, decimals=3):
    print("Evaluation counts:", evaluation_ratios_counts)
    try:
        positive_predictive_value = evaluation_ratios_counts["TP"]/(evaluation_ratios_counts["TP"]+evaluation_ratios_counts["FP"])
    except:
        positive_predictive_value = 0
    try:
        true_positive_rate = evaluation_ratios_counts["TP"]/(evaluation_ratios_counts["TP"]+evaluation_ratios_counts["FN"])
    except:
        true_positive_rate = 0
    try:
        false_discovery_rate = evaluation_ratios_counts["FP"]/(evaluation_ratios_counts["TP"]+evaluation_ratios_counts["FP"])
    except:
        false_discovery_rate = 0
    try:
        false_positive_rate = evaluation_ratios_counts["FP"]/(evaluation_ratios_counts["FP"]+evaluation_ratios_counts["TN"])
    except:
        false_positive_rate = 0
    try:
        false_omission_rate = evaluation_ratios_counts["FN"]/(evaluation_ratios_counts["TN"]+evaluation_ratios_counts["FN"])
    except:
        false_omission_rate = 0
    try:
        false_negative_rate = evaluation_ratios_counts["FN"]/(evaluation_ratios_counts["TP"]+evaluation_ratios_counts["FN"])
    except:
        false_negative_rate = 0
    try:
        negative_predictive_value = evaluation_ratios_counts["TN"]/(evaluation_ratios_counts["TN"]+evaluation_ratios_counts["FN"])
```

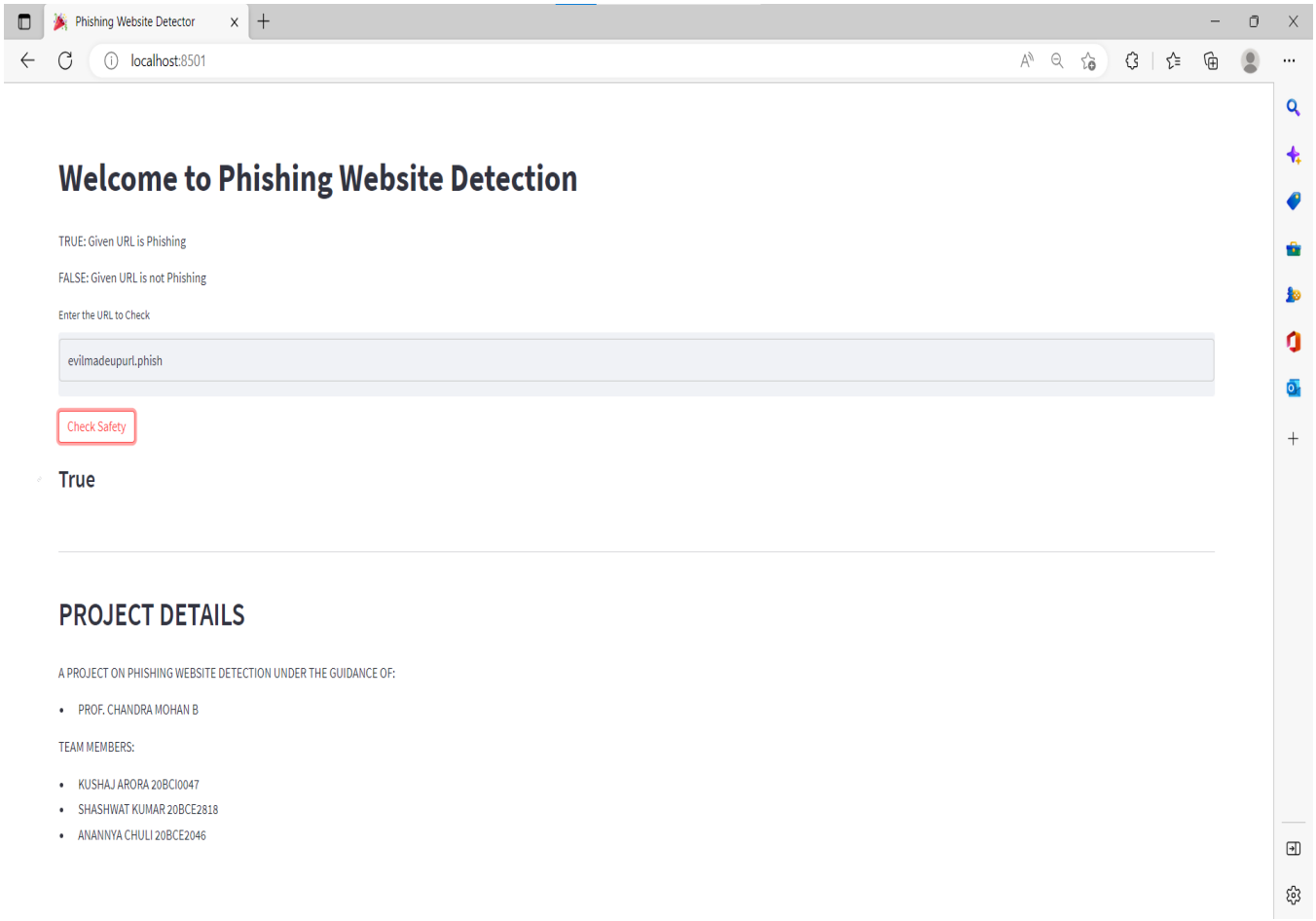
```
    except:
        accuracy = 0
    t = PrettyTable([f"Accuracy {np.round(accuracy*100, decimals=decimals)}%",
                     'Predicted safe',
                     'Predicted phishing'])
    t.add_row(["Not phishing",
               "TN: {TN}".format(**evaluation_ratios_counts),
               "FP: {FP}".format(**evaluation_ratios_counts)])
    t.add_row(['', f"NPV: {np.round(negative_predictive_value*100, decimals=decimals)}%",
                   f"FDR: {np.round(false_discovery_rate*100, decimals=decimals)}%"])
    t.add_row(['', f"TNR: {np.round(true_negative_rate*100, decimals=decimals)}%",
                   f"FPR: {np.round(false_positive_rate*100, decimals=decimals)}%"])
    t.add_row(["Is phishing",
               "FN: {FN}".format(**evaluation_ratios_counts),
               "TP: {TP}".format(**evaluation_ratios_counts)])
    t.add_row(['', f"FOR: {np.round(false_omission_rate*100, decimals=decimals)}%",
                   f"PPV: {np.round(positive_predictive_value*100, decimals=decimals)}%"])
    t.add_row(['', f"FNR: {np.round(false_negative_rate*100, decimals=decimals)}%",
                   f"TPR: {np.round(true_positive_rate*100, decimals=decimals)}%"])
    print(t)

def threshold_evaluation_plotter(X, y, min_threshold=0.05, max_threshold=0.95, steps=200, decimals=3):
    predictions = model.predict(X).flatten()
    stat_counts = []
    # Sweeping over the ranges.
    for threshold in np.linspace(min_threshold, max_threshold, steps):
        predictions_boolean = predictions > threshold
        predictions_binary = predictions_boolean.astype(np.int)
        evaluation_ratios_counts, sample_outcomes = statistics_evaluator(predictions_binary, y)
        stat_counts.append(evaluation_ratios_counts)
    counts_df = pd.DataFrame(data=stat_counts, index=np.linspace(min_threshold, max_threshold, steps))
    stat_df = pd.DataFrame(index=np.linspace(min_threshold, max_threshold, steps))
    # Calculating the stats:
    stat_df["accuracy"] = (counts_df["TP"]+counts_df["TN"])/(counts_df["TP"]+counts_df["TN"]+counts_df["FP"]+counts_df["FN"])
    stat_df["PPV"] = counts_df["TP"]/(counts_df["TP"]+counts_df["FP"])
    stat_df["TPR"] = counts_df["TP"]/(counts_df["TP"]+counts_df["FN"])
    stat_df["FDR"] = counts_df["FP"]/(counts_df["TP"]+counts_df["FP"])
    stat_df["FPR"] = counts_df["FP"]/(counts_df["FP"]+counts_df["TN"])
    stat_df["FOR"] = counts_df["FN"]/(counts_df["TN"]+counts_df["FN"])
    stat_df["FNR"] = counts_df["FN"]/(counts_df["TP"]+counts_df["FN"])
    stat_df["NPV"] = counts_df["TN"]/(counts_df["TN"]+counts_df["FN"])
    stat_df["TNR"] = counts_df["TN"]/(counts_df["FP"]+counts_df["TN"])
    fig = stat_df.plot(kind='line',  figsize=(20, 7), fontsize=16, lw=3).get_figure()
```

```
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    print('gpu', gpu)
    tf.config.experimental.set_memory_growth(gpu, True)
    print('memory growth:' , tf.config.experimental.get_memory_growth(gpu))

# https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.LSTM(96),
    tf.keras.layers.Dense(128,activation="tanh"),
    tf.keras.layers.Dense(1, activation="sigmoid")
])


# Compiling the model
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=['acc'])
print(model.summary())

class_weight={0: (1/(oversampling_rate+1)), 1: (oversampling_rate/(oversampling_rate+1))}
print("Using the class weighting:", class_weight)
# Training the model
# Setting up callback to monitor the selected loss, and stops training if it doesnt improve for patience-number of epo
# After stopping training will restore the weights from the best iteration on this value encountered so far.
early_stopping_callback = tf.keras.callbacks.EarlyStopping(monitor="val_acc", patience=4, restore_best_weights=True)
history = model.fit(X_train_encoded_padded, y_train,
                    epochs=100,
                    validation_data=(X_test_encoded_padded, y_test),
                    class_weight=class_weight,
                    sample_weight=sample_weights_train,
                    callbacks=[early_stopping_callback])

# Model built in evaluate
results = model.evaluate(X_test_encoded_padded, y_test)
print(results)

# Custom evaluate
best_threshold = threshold_evaluation_plotter(X_test_encoded_padded, y_test)
mean_prediction = evaluate_nn_model(X_test_encoded_padded, y_test, threshold=best_threshold)

# Saving the model:
model.save('model.h5')
```

OUPTUT SNIPPETS:







LINK:

http://phishin-detection.herokuapp.com/

CONCLUSION

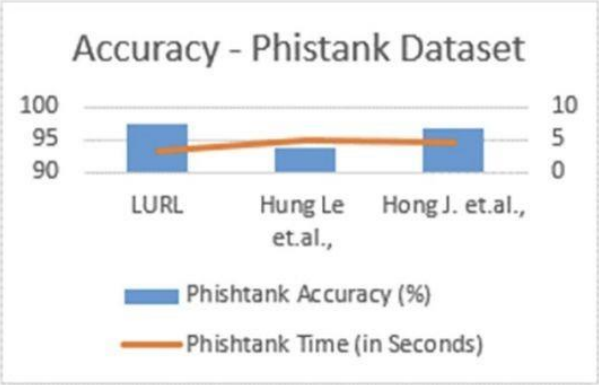
Comparison of Accuracy

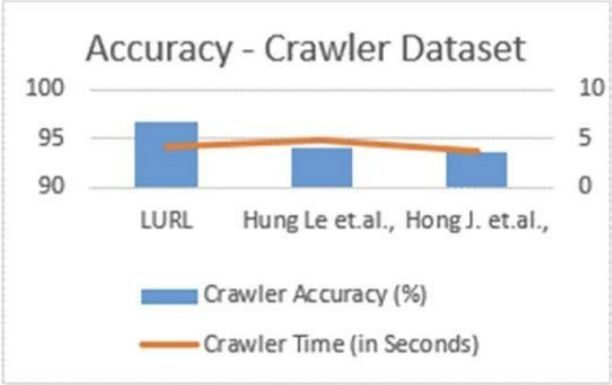
Methods	Phishtank		Crawler	
	Accuracy (%)	Time (in Seconds)	Accuracy (%)	Time (in Seconds)
LURL	97.4	3.45	96.8	4.21
Hung Le et al.	93.8	5.12	94.1	4.79
Hong J. et al.	96.7	4.78	93.6	3.79

Comparison of F1- Measure

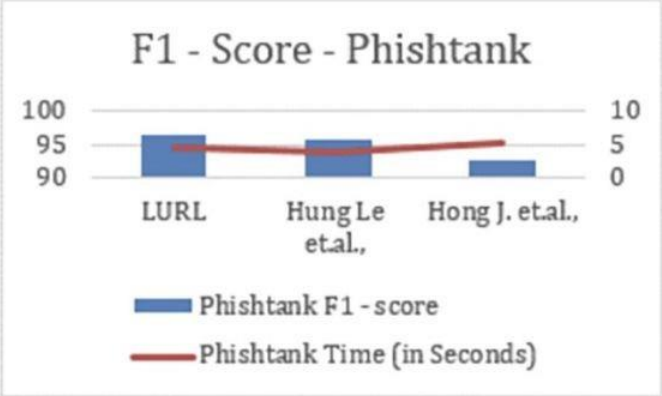
Methods	Phishtank		Crawler	
	F1- Score	Time (in Seconds)	F1–Score	Time (in Seconds)
LURL	96.4	4.62	96.2	3.89
Hung Le et al.	95.8	3.87	95.6	4.91
Hong J. et al.	92.7	5.23	95.3	3.62

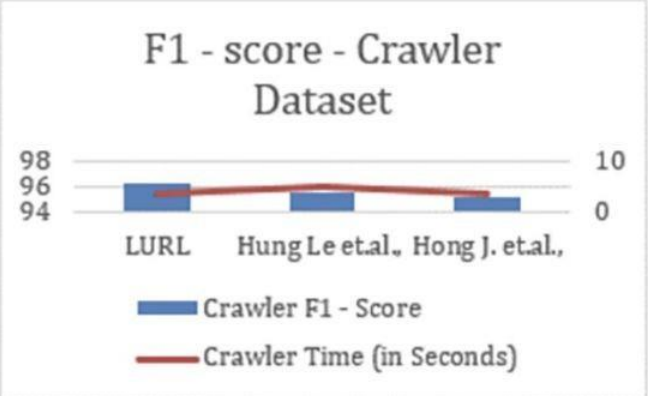
The above table provides the comparison of F1—score of URL detectors. As presented in section 2, TP and TN indicate the malicious and legitimate URLs, accordingly. Based on the TP, TN, FP, and FN, both precision and recall value are calculated. Using these values, F1— measure is computed. It indicates the retrieving ability of URL detector. From the outcome, it is obvious that the proposed URL detector, LURL is superior rather than other two URL detectors. The reason for the better F1—measure is the capability of LSTM memory





The above graph represents that LURL has generated the output in less amount of time rather than the other predictors.





The above graph shows the F1—score against the computation time.

It represents that LURL achieved a F1—Scoreof 96.4 in 4.62 seconds for Phishtank dataset.

Learning Rate	LURL	Hung Le et al.	Hong J. et al.
1.0	87.5	86.8	89.7
2.0	89.2	88.1	89.6
3.0	90.6	91.3	90.3
4.0	91.7	90.8	91.8
5.0	93.6	92.4	94.1

Crawler dataset contains larger number of normal URLs comparing to the malicious URLs. The intention for employing Crawler is to teach the methods to predict legitimate URLs. It is very difficult to predict a website without analyzing content; however, the phishing site is similar to legitimate website.

Therefore, it is necessary for methods to understand the differences between legitimate and malicious website. it is obvious that the performance of all detectors is like each other. Similar to Phish tank dataset, all three methods consumed an average of 86% of data at the rate of 1.0. The reason for the faster rate is that RNN can read numeric data at faster than images.

LURL has produced an average of 97.4% and 96.8% for Phish tank and Crawler datasets respectively. Both Hung Le et al., and Hong J. et al., have reached an average of 93.8, 94.1, 96.7, and 93.6 for Phish tank and Crawler datasets. It is evident that the performance of LURL is better comparing to other URL.

The proposed solution of this project emphasized the phishing technique in the context of classification.

- The phishing website is considered to involve automatic categorization of websites into a predetermined set ofclass values based on several features and the class variable.
- The ML based phishing techniques depend on website functionalities to gather information that can help classify websites for detecting phishing sites.
- The problem of phishing cannot be eradicated, nonetheless can be reduced by combating it in two ways, improving targeted anti-phishing procedures and techniques and informing the public on how fraudulent phishing websites can be detected and identified.
- To combat the ever evolving and complexity of phishing attacks and tactics, ML anti-phishing techniques are essential.

- Authors employed LSTM technique to identify malicious and legitimate websites.
- A crawler was developed that crawled 7900 URLs from AlexaRank portal and also employed Phishtankdataset to measure the efficiency of the proposed URL detector.
- The outcome of this study reveals that the proposed method presents superior results rather than theexisting deep learning methods.
- A total of 7900 malicious URLS were detected using the proposed URL detector.
- It has achieved better accuracy and F1—score with limited amount of time.

The future direction of this study is to develop an unsupervised deep learning method to generate insight from a URL. In addition, the study can be extended in order to generate an outcome for a larger network and protect the privacy of an individual.

REFERENCES

- Gandotra E., Gupta D, “An Efficient Approach for Phishing Detection using Machine Learning”, Algorithms for Intelligent Systems, Springer, Singapore, 2021.
- Hung Le, Quang Pham, Doyen Sahoo, and Steven C.H. Hoi, “URLNet: Learning a URL Representation withDeep Learning for Malicious URL Detection”, Conference’17, Washington, DC, USA, arXiv:1802.03162, July 2017.
- Srinivasa Rao R, Pais AR, “Detecting phishing websites using automation of human behavior”, In: Proceedings of the 3rd ACM workshop on cyber-physical system security, ACM, pp 33–42, 2017.
- Sahingoz OK, Buber E, Demir O, Diri B, “Machine learning based phishing detection from URLs”, Expert System Application, vol. 117, pp. 345–357, 2019.
- Zamir A, Khan HU, Iqbal T, Yousaf N, Aslam F et al., “Phishing web site detection using diverse machine learning algorithms”, The Electronic Library, vol.38, no.1, pp. 65–80, 2019
- Rao RS, Pais AR. Jail-Phish: An improved search engine based phishing detection system. Computers & Security. 2019 Jun 1; 83:246–67
- Feature Selection for Phishing Website Classification. Int. J. Adv. Comput. Sci. Appl. Shabudin, S.; Sani,N.S.; Ariffin, K.A.Z.; Aliff, M. (2020).
- Phishing Attacks and Websites Classification Using Machine

Learning and Multiple Datasets - Khan, S.A.Khan, W. Hussain (2020).

- Phishing page detection via learning classifiers from page layout feature Jian Mao, Jingdong Bian,Wenqian Tian, Shishi Zhu Tao Wei, Li and Zhenkai Liang (2019).
- Malicious web pages detection using feature selection techniques and machine learning. Int. J. High.Perform. Comput. Netw. Patil, D.R.; Patil, J.B. (2019).
- Intelligent phishing website detection using random forest classifier. Proceedings of the 2017 InternationalConference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, United Arab Emirates.Patil, D.R.; Patil, J.B. (2017).
- Hybrid intelligent phishing website prediction using deep neural networks with genetic algorithm-based feature selection and weighting - Ali, W.; Ahmed, A.A. (2019).
- A new hybrid ensemble feature selection framework for machine learning-based phishing detection system - Chiew, K.L.; Tan, C.L.; Wong, K.; Yong, K.S.; Tiong, W.K. (2019).
- Ali, Waleed. "Phishing website detection based on supervised machine learning with wrapper features selection." International Journal of Advanced Computer Science and Applications 8.9 (2017)
- Phishing page detection via learning classifiers from page layout feature Jian Mao, Jingdong Bian, Wenqian Tian, Shishi Zhu Tao Wei, Li and Zhenkai Liang (2019).