Detecting Malicious Communications: An Investigation into One-Hot Encoding and CNN-RNN Model for SMS Spam and phishing Email Classification

Asim, Muhammad Muwahid*
Faculty of Computer Science and Engineering
Ghulam Ishaq Khan Institute
Topi, Pakistan
asim.muwahid@gmail.com

Khan, Muhammad Ramish*
Faculty of Computer Science and Engineering
Ghulam Ishaq Khan Institute
Topi, Pakistan
kramish241@gmail.com

Khan, Farhan
Faculty of Computer Science and Engineering
Ghulam Ishaq Khan Institute
Topi, Pakistan
khan,farhan@giki.edu.pk

Khan, Abdul Basit
Faculty of Computer Science and Engineering
Ghulam Ishaq Khan Institute
Topi, Pakistan
u2019004@giki.edu.pk

Qureshi, Muhammad Saaim
Faculty of Computer Science and Engineering
Ghulam Ishaq Khan Institute
Topi, Pakistan
u2019444@giki.edu.pk

Hou, Zhe
School of Information and Communication Technology
Griffith University
Queensland, Australia
z.hou@griffith.edu.au

***Abstract*—This paper 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. We implemented our proposed approach for phishing email detection. The results of our 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. Our 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 we 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.**

***Index Terms*—deep learning, artificial intelligence, cyberattacks, character-wise one-hot encoding, convolutional neural network, recurrent neural network, phishing email detection, effectiveness, risks mitigation, accuracy, area under the receiver operating characteristic curve, AUC score, SMS spam detection, reducing risks.**

I. Introduction

This paper 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. 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.

The surge in phishing attacks poses a growing risk to online identity security, leading to substantial financial losses and compromising the safety of commercial transactions on the Internet. As businesses increasingly operate in an online environment, the need for effective defenses against identity theft, fraud, malware, and phishing attacks becomes paramount.

Machine learning, particularly in the era of big data, has emerged as a powerful tool in addressing these challenges. Big data enables machine learning algorithms to discern nuanced patterns and make precise predictions. Deep learning techniques, characterized by non-linear operations in multiple levels, including neural networks with hidden layers, have proven effective in various applications such as object identification, voice-to-text transcription, and personalized content matching.

The Long Short-Term Memory (LSTM) network, recognized for its excellence in character recognition, handwriting recognition, speech recognition, and other applications, stands

out as a promising tool. Recent studies propose models utilizing URL and HTML features, leveraging lightweight features for real-time phishing detection with high accuracy rates. Text feature-based approaches, including the use of LSTM, have also demonstrated efficiency in identifying phishing attacks.

The research not only emphasizes the significance of deep learning and AI models in countering cyber threats but also showcases the ongoing efforts to tackle evolving challenges, such as phishing attacks.

II. Related Work

Li et al. (2019) proposed a model based on URL and HTML features to detect phishing web pages. They also designed lightweight features of HTML and URL, which they developed as HTML string-embedding features without using third-party services, which allows their model to work in a real-time detection application. They tested their method on a real-life data set that consisted of over 100,000 URL and HTML features. The authors reported that their scheme was able to achieve 97.30% accuracy, a 4.46% true positive rate and a 1.61% false negative rate (Li et al., 2019). Mishra and Gupta (2018) also focussed on the text features of websites in order to propose a novel system for intelligent phishing detection to detect zeroday phishing attacks. For their model, the authors used the concept of uniform resources identifier and cascading style sheet matching. The authors reported that the proposed solution is very efficient in detecting phishing and zero-day attacks with a true positive rate of 93.27% (Mishra and Gupta, 2018). The aforementioned studies used the text features of websites to develop their phishing detection models. However, phishers can use other content on sites to evade detection.

Bahnsen et al. (2017) investigated the performance of LSTM in their work on a solution for phishing site prediction that uses URLs as input for machine learning models. The authors compared a feature engineering approach with the random forest (RF) classifier against a novel method based on an RNN. They used 14 features to build their lexical and statistical analysis of the URLs. They used an LSTM unit to build the model that receives as input a URL as a sequence of characters and predicts whether the URL is phishing or legitimate. They also constructed a data set that consisted of 2m phishing and legitimate URLs to train their model. They found that the LSTM model had an overall higher prediction accuracy compared to the RF classifier without the need for expert knowledge to create the features. Their approach was able to achieve an accuracy rate of 98.7% even without manual feature creation (Bahnsen et al., 2017). However, again, their study only focused on the text based URLS.

The use of Convolutional Neural Networks (CNNs) for text feature extraction has gained significant attention in recent years. One-dimensional CNNs have been successfully employed to capture local patterns and dependencies within sequential data, such as text. These models leverage the power of convolutional filters to extract meaningful features at different levels of abstraction. For instance, Kim (2014) introduced a one-dimensional CNN architecture for sentence classification, demonstrating the effectiveness of such models in capturing syntactic and semantic information. Similarly, Zhang et al. (2015) proposed a character-level CNN for text classification, achieving state-of-the-art results on various benchmark datasets. These works paved the way for the application of one-dimensional CNNs in text-related tasks, showcasing their potential in extracting relevant features from textual data.

Y. Fang et al proposed a method of using email body to identify the phishing attempts. The paper introduces a novel phishing email detection model named THEMIS, which leverages an improved recurrent convolutional neural networks (RCNN) model with multilevel vectors and an attention mechanism. The proposed model simultaneously models various aspects of emails, including the header, body, character level, and word level. However, there is a increased number of trainable parameters resulting from the use of multi-LSTM layers in the proposed model.

III. Technical Background

All vectors used in this section are column vectors denoted by boldface lowercase letters. Matrices are denoted by boldface uppercase letters. For a vector $\mathbf{x}$ (or a matrix vU), $\mathbf{x}^T$ ($\mathbf{U}^T$) is the ordinary transpose. N is the total number of time steps, and an arbitrary time-step is denoted by $\mathbf{t}$ where $0 \leq$ t $\leq$ N - 1. The time index of a sequence vector is denoted by t in the subscript, as in $\mathbf{x}^T$.

We represent the input (a representation of the payloads) as column vectors of a matrix $X[n]$, where each column vector is a sequence vector $x_t[n]$ at time t for sample n, i.e., $X[n] = [x_0[n]x_1[n]... \ x_{N-1}[n]]$. We denote the desired output by a scalar $y[n] \in 0, 1$, where 0 represents ham and 1 represents anomaly or phishing email.

For phishing email detection, we use a deep learning framework to analyze the contents of the email. We consider each email as a document and use an encoding method with a series of CNNs that extracts relevant features. The framework is trained on a dataset of legitimate and phishing emails to learn the patterns and characteristics of each type. We use a sequential learning approach to analyze the email contents in a sequential manner, character by character, to identify any suspicious patterns. Additionally, we investigate a mechanism to extract contextual information by incorporating the grammar and syntax of the email. This enables us to identify any anomalies in the email structure that may indicate phishing attempts. Overall, our framework uses a combination of CNNs and RNNs to accurately detect phishing emails and protect users from potential cyber-attacks.

A. Feed forward neural network (FFN)

Feed forward neural network (FFN) forms a directed graph which is composed of nodes and edges. FFN passes information along edges from one node to another without forming a cycle. Multi-layer perceptron (MLP) is one type of

feed forward neural network that contains 3 or more layers, specifically input layer, one or more hidden layer and an output layer in which each layer has many neurons, called as units in mathematical notation. The number of hidden layers is selected through following fine-tuning mechanism. The information is transformed from one layer to another layer in forward direction without considering the past values. Moreover, neurons in each layer are fully connected. MLP is defined mathematically as $O : R^m X R^n$ where m is the size of the input vector $x = x_1, x_2, x_3 \dots x_m$ and n is the size of the output vector $O(x)$ respectively. The computation in each hidden layer hi_i is mathematically defined as

$$hi_i = f\left(w^T x + b_i\right) \tag{1}$$

$$hi_i : R^{d_i - 1} \rightarrow R^{d_i}, f : R \rightarrow R, \text{ where } w \in R^{d \times d_{i-1}},$$

b $\in R^d d_i$ denotes the size of the input, f is non-linear activation function, either sigmoid (values in the range $[0, 1]$) or tangent function (values in the range $[1, -1]$).

$$\sigma(x) = \frac{1}{1 + \exp(-z)} \tag{2}$$

B. Convolutional neural network (CNN)

For an input sequence $X = [x_0 x_1 ... \ x_{T-1}]$, we use a multiple layer CNN.

Here, the output of a CNN unit is given as

$$y_{i,j} = \Sigma_{k=j-r+2}^{j+r-2} \boldsymbol{w}_{i,j,k}^T \boldsymbol{x}_k \tag{3}$$

where r is the order of the convolution unit, i is the index of the layer, j is the index of the unit inside the layer, and T is the number of input sequence vectors.

The first convolution layer of order r, i.e., $i = 1$, takes an input X comprising of the sequence vectors $x[0]$, $x[1]$, ..., $x[T-1]$ and produces scalar outputs $y_{1,0}$, $y_{1,1}$, ..., $y_{1,T-r}$ or a vector $\boldsymbol{y}_1 \in \mathbb{R}^{T-r+1}$.

The weights $w_{i,j,k} \in 0, 1$ are d-dimensional vectors. We use F such convolution filters at each layer, hence resulting in a $(T - r + 1) \times F$ dimensional matrix or array. The process is repeated L times and hence reducing the number of rows at each layer. We also use p order pooling at intermediate levels, between the layers, i.e.,

$$\mathbf{o} = \text{pool}\left(y_{I,l}\right), \tag{4}$$

where pool(.) is either average pooling, i.e.,

$$\text{pool}\left(z_l\right) = \frac{1}{L} \Sigma_{l=1}^{L} z_l, \tag{5}$$

or max pooling, i.e.

$$\text{pool}\left(z_l\right) = \max_{l=1}^{L}\left(z_l\right), \tag{6}$$

and L is the number of inputs to the pooling unit.

We consider an input sequence,, $\boldsymbol{X} \in \mathbb{R}^{d \times T}$ where $x_t \in X$ is a dimensional sequence vector such that $X = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \dots \boldsymbol{x}_{T_i - 1}\}$.

We use a L layer convolution network, each comprised of $T_i - r_{i-1} - 1$ units where r_i is the order of the convolution unit and T_i is the number of inputs at layer i. We pass the final output $\tilde{X} \in \mathbb{R}^{P \times F}$ to the LSTM network for payload and connection-level encoding, such that the temporal input is input is $\tilde{\boldsymbol{x}}_t \in \mathbb{R}^P$ where $0 \leq t \leq F - 1$.

C. Recurrent neural network (RNN)

For the sequential and structural learning, we use RNNs that extract the sequential information and to some extent the contextual information from the input.

1) long short-term memory networks (LSTM): We use LSTM units to build RNN models. LSTM-based RNNs are useful for long-term dependencies since they connect the previous information as well as the current input to the current task. LSTMs are long chains of memory units where each unit takes the input at time t, the information gathered from the previous units till time $t-1$, and generates a candidate for the output as well as a state to be passed to the next unit.

The LSTM has an input $x(t)$, which can be the output of a CNN or the input sequence directly. $h(t-1)$ and $c(t-1)$ are the inputs from the previous timestep LSTM. $o(t)$ is the output of the LSTM for this timestep. The LSTM also generates $c(t)$ and $h(t)$ for the consumption of the next timestep LSTM.

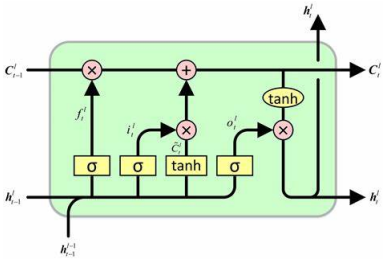

Fig. 1. One cell of LSTM

$$f_t = \sigma_g\left(W_f \times x_t + U_f \times h_{t-1} + b_f\right) \tag{7}$$

$$i_t = \sigma_g\left(W_i \times x_t + U_i \times h_{t-1} + b_i\right) \tag{8}$$

$$o_t = \sigma_g\left(W_o \times x_t + U_o \times h_{t-1} + b_o\right) \tag{9}$$

$$c'_t = \sigma_c\left(W_c \times x_t + U_c \times h_{t-1} + b_c\right) \tag{10}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t \tag{11}$$

$$h_t = o_t \cdot \sigma_c\left(c_t\right) \tag{12}$$

$where$,

f_t is the forget gate

i_t is the input gate

o_t is the output gate

c_t is the cell state

h_t is the hidden state

$\sigma_c : sigmoid$

$\sigma_g : tanh$

. : element-wise multiplication

The weight matrices W_f, W_i, W_c, W_o, and bias vectors b_f, b_i, b_c, and b_o are updated using back propagation during training, at each sample input.

2) GRU: GRU based RNNs are also effective for handling long term dependencies in sequential data. Similar to LSTM, GRU units can capture information from the previous inputs and carry them forward to the current task. GRU units consist of memory gates that control the flow of information, allowing them to selectively forget or retain information from previous time steps. In particular, the GRU unit has a reset gate and an update gate that controls the flow of information, allowing it to selectively forget or retain information from previous time steps. This helps in avoiding the vanishing gradient problem that can arise in traditional RNNs. Overall, the use of GRU units in RNNs can lead to improved performance in sequential learning tasks by capturing the long-term dependencies in the data.

The GRU also has an input $x(t)$ which can be the output of a CNN or the input sequence directly. Additionally, it takes the hidden state from the previous time step $h(t-1)$ as input. The GRU generates an update gate $z(t)$, a reset gate $r(t)$, and a new candidate hidden state $\tilde{h}(t)$. These gates determine the information that is retained from the previous time step and the information that is added to the current hidden state. Finally, the GRU outputs the current hidden state $h(t)$, which can be used for classification or consumption of the next time step GRU.

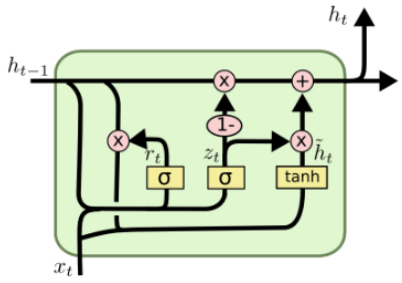

Fig. 2. One cell of GRU

$$z = \sigma\left(W_z \times x_t + U_z \times h_{t-1} + b_z\right) \tag{13}$$

$$r = \sigma\left(W_r \times x_t + U_r \times h_{t-1} + b_r\right) \tag{14}$$

$$\tilde{h} = tanh\left(W_h \times x_t + r * U_h \times h_{t-1} + b_z\right) \tag{15}$$

$$h = z * h_{t-1} + (1 - z) * \tilde{h} \tag{16}$$

Similarly, the weight matrices W_z, W_r, and W are updated by using backpropagation at each sample input during training. The bias vectors b_z, b_r, and b are also updated accordingly.

In a single layer RNN network, the output h_t of the last LSTM unit is further passed through a logistic regression unit to estimate the final output, i.e., the label of the email whether phishing or not. In principle, the RNN network turns a matrix input $X[n]$ to a feature vector $h_{N-1}[n]$ that is used in a classic machine learning problem. However, in practice, multiple layers of RNNs are used to produce the final output. In such a case, all the outputs of the previous layers are used as input to the next layer, and in the final layer, only the last output is used.

RNNs learn the inherent sequential model as well as the contextual and syntactic information since they incorporate memory from the sequential inputs. However, to make the model further robust, specifically to learn the character-level encoding and the mutual information, we add CNNs on top of LSTM RNNs.

IV. Proposed Approach

Let $E = e_1, e_2, \ldots, e_n$ be a set of email samples and $C = c_1, c_2, \ldots, c_n$ be a set of email types such as legitimate or phishing, where n denotes the number of email samples. The task is to classify each given email sample into either legitimate or phishing.

A. Preprocessing

We use one-hot encoding to convert each character in the input sequence to a 98-dimensional vector. The payloads can have varying lengths, up to 8000 characters, and we limit the length of payloads to L = 4000 by taking their average. If the length of a payload is less than L, i.e., N $\leq$ L-1, we pad the one-hot encoded output with (L - N) vectors comprising of -1's. Then, we use a masking layer in the deep learning model so that when the input is -1, the weights are not updated. This way, each sample input is comprised of L 98-dimensional arrays.

B. Proposed Architecture of Deep Neural Network

The proposed architecture to identify phishing emails is shown in Figure 1. The architecture consists of the following modules:

Representation of email: In the proposed architecture, instead of utilizing word2vec or bag-of-ngrams, a one-hot character-wise encoding scheme is employed to represent the emails. This approach represents each character in the email as a one-hot vector, indicating its presence or absence. The encoding scheme retains the sequential information of the email text.

To implement the one-hot character-wise encoding, each character in the email is mapped to a vector of length 98. Each position in the vector corresponds to a specific character, and a value of 1 is assigned to the position that represents the corresponding character, while all other positions are set to 0. This one-hot encoding captures the presence or absence of each character in the email.

Deep learning: The one-hot encoded character vectors are passed as input to the CNN and RNN networks. These networks capture the appropriate feature representations and pass them into fully connected layers for classification. The detailed configuration details of the CNN and RNN architectures are reported in Table 1. The first layer in the architecture is the masking layer, which ensures that the model does not train on the "-1" task.

Classification: The units in this layer have connection to every other unit in the succeeding layer. That's why this layer is called as fully-connected layer. It contains sigmoid nonlinear activation function, which gives values 0 for legitimate and 1 for phishing. The prediction loss for both the sub tasks is estimated using binary cross entropy.

C. Evalution metrics

In the proposed architecture for email classification, several evaluation metrics are employed to assess the performance of the model. These metrics include the area under the ROC curve (AUC score), accuracy, and precision.

The AUC score, or area under the receiver operating characteristic curve, is a widely used metric for evaluating binary classification models. It measures the ability of the model to distinguish between positive and negative samples by assessing the trade-off between true positive rate and false positive rate. A higher AUC score indicates a better-performing model with improved discrimination power.

Accuracy is a straightforward metric that measures the overall correctness of the model's predictions. It calculates the ratio of correctly classified samples to the total number of samples. While accuracy provides a general overview of the model's performance, it may not be suitable when the dataset is imbalanced.

Precision, on the other hand, focuses on the correct prediction of positive samples. It measures the ratio of true positives to the sum of true positives and false positives. Precision is particularly useful when the cost of false positives is high, as it emphasizes the avoidance of miss-classifying legitimate emails as phishing.

These evaluation metrics collectively provide insights into different aspects of the model's performance. The AUC score assesses the discriminatory power, accuracy provides an overall measure of correctness, and precision highlights the ability to correctly classify positive samples. By considering these metrics, one can comprehensively evaluate the effectiveness of the proposed architecture for email classification.

D. Algorithm

Algorithm 1 presented is a pseudocode representation of a machine learning model designed for training on email content data, likely for a binary classification task such as spam detection. The input to the algorithm is the content of an email, and the desired output is denoted as 'out.' The training process involves several steps.

Firstly, the email content undergoes preprocessing, which includes one-hot encoding and padding to create a suitable input format for the subsequent layers. The Masking layer is then applied to handle sequences with varying lengths, where a mask value of -1 is utilized during the padding process.

Next, Convolutional Neural Network (CNN) layers are introduced to capture local patterns within the email content. Multiple filters of different sizes are applied to the masked and padded email body, and the resulting convolutional outputs are passed through a rectified linear unit (ReLU) activation function. Pooling is applied to downsample the obtained features.

Following the CNN layers, Recurrent Neural Network (RNN) layers are employed to capture sequential dependencies in the data. Different types of RNNs (as specified by 'rnnType') and their corresponding numbers of units are iteratively applied to the pooled outputs from the CNN layers.

Subsequently, Dense layers are introduced to further learn complex representations from the combined CNN and RNN features. The number of neurons in each dense layer is determined by the parameter 'numNeurons.'

Finally, the algorithm concludes with an output layer that produces the final prediction. The output is a binary classification, and the activation function used is the sigmoid function, indicating a probability-like output. The overall architecture of the model combines convolutional, recurrent, and dense layers to effectively learn hierarchical representations from the email content, ultimately enabling it to make predictions about the nature of the email (e.g., spam or not spam).The algorithm outputs a value (denoted as 'output') representing the predicted probability of the input email belonging to the positive class (e.g., being spam).

Algorithm 2 represents a pseudocode for the inference phase of a machine learning model designed for predicting the nature of email content, likely for a binary classification task such as spam detection. The input to the algorithm is the content of an email, and the output is denoted as 'prediction.'

The inference process begins with the same preprocessing steps as the training phase, where the email content is one-hot encoded and padded to create a suitable input format. The Masking layer is applied to handle varying sequence lengths, utilizing a mask value of 1 during the padding process.

Subsequently, the algorithm applies Convolutional Neural Network (CNN) layers to capture local patterns in the email content. Multiple filters of different sizes are used, and the resulting convolutional outputs are passed through a rectified linear unit (ReLU) activation function. Pooling is then applied to downsample the obtained features.

Following the CNN layers, Recurrent Neural Network (RNN) layers are employed to capture sequential dependencies in the data. Different types of RNNs (specified by 'rnnType')

and their corresponding numbers of units are iteratively applied to the pooled outputs from the CNN layers.

Next, Dense layers are introduced to further learn complex representations from the combined CNN and RNN features. The number of neurons in each dense layer is determined by the parameter 'numNeurons.'

The algorithm concludes with an output layer that produces the final prediction. The output is a binary classification, and the activation function used is the sigmoid function, indicating a probability-like output. The 'output' variable represents the predicted probability of the input email belonging to the positive class (e.g., being spam). The algorithm then returns this probability as the 'prediction.' This inference algorithm leverages the previously trained model architecture to make predictions on new, unseen email content during the deployment phase.

CNN Layers	RNN Layers	Trainable Parameters	Loss	Accuracy	Precision	AUC Score
2: 1dconv(64 filters) 1dconv(64 filters)	2: LSTM (64 filters) LSTM (32 filters)	150,401	0.353	0.872	0.806	0.876
2: 1dconv(128 filters) 1dconv(64 filters)	Bidirectional LSTM(256 filters)	309,761	0.602	0.866	0.800	0.844
2: 1dconv(128 filters) 1dconv(64 filters)	2: Bidirectional LSTM(256 filter) Bidirectional LSTM (128 filters)	470,017	0.347	0.871	0.806	0.864
2: 1dconv(128 filters) 1dconv(64 filters)	2: Bidirectional LSTM(256 filter) Bidirectional LSTM (128 filters)	486,529	0.873	0.867	0.808	0.851
3: 1dconv(128 filters) 1dconv(64 filters) 1dconv(32 filters)	3 : LSTM (64 filters) LSTM (32 filters) LSTM (32 filters)	148,385	0.333	0.879	0.812	0.876
3 : 1dconv(128 filters) 1dconv(64 filters) 1dconv(32 filters)	4: LSTM (128 filters) LSTM (64 filters) LSTM (32 filters) LSTM (32 filters)	263,713	0.334	0.875	0.814	0.878
3 : 1dconv(128 filters) 1dconv(64 filters) 1dconv(32 filters)	2: GRU (64 filters) GRU(32 filters)	139,361	0.334	0.876	0.810	0.876
2: 1dconv(128 filters) 1dconv(64 filters)	GRU(32 filters)	113,313	0.357	0.870	0.812	0.873
3 : 1dconv(128 filters) 1dconv(128 filters) 1dconv(128 filters)	3: GRU(128 filters) GRU (64 filters) GRU(32 filters)	291,841	0.333	0.876	0.811	0.876

Table 1. Comparison of Results

Algorithm 1 Pseudo code for training

Input: email content
Output: out
*p*reprocessing :
OneHotEncodedEmailBody = oneHotEncode(emailBody, 98)
PaddedEmailBody = pad(OneHotEncodedEmailBody, L)
*M*asking layer :
maskedEmailBody = Masking(mask_value= -1)(padded EmailBody)
*C*NN Layers:
for each *filter_size*, *num_filters* in *cnn_layers* **do**
    ConvOutput = Conv1D(NumFilters, FilterSize, activation='relu')(MaskedEmailBody)
    poolOutput=Pooling1D(poolSize=POOLSIZE)(ConvOutput)
end for
*R*NN Layers :
for each *rnnType*, *numUnits* in *rnn_layers* **do**
    rnnOutput = rnnType(numUnits)(poolOutput)
end for
*D*ense Layers :
for each *numNeurons* in *dense_layers* **do**
    denseOutput=Dense(1,activation='sigmoid')(rnnOutput)
end for
*O*utput Layer :
output = Dense(1, activation='sigmoid')(denseOutput)
*O*utput :
return $output$ =0

Algorithm 2 Pseudo code for inference

Input: email content
Output: prediction
OneHotEncodedEmailBody = oneHotEncode(emailBody, 98)
PaddedEmailBody = pad(OneHotEncodedEmailBody, L)
Masking layer:
maskedEmailBody = Masking(mask_value=-1)(PaddedEmailBody) *CNN Layers*:
for each *filter_size*, *num_filters* in *cnn_layers* **do**
    ConvOutput = Conv1D(numFilters, filterSize, activation='relu')(maskedEmailBody)
    PoolOutput = Pooling1D(poolSize=POOLSIZE)(ConvOutput)
end for*RNN Layers*:
for each *rnnType*, *numUnits* in *rnn_layers* **do**
    RNNOutput = rnnType(numUnits)(PoolOutput)
end for*Dense Layers*:
for each *numNeurons* in *dense_layers* **do**
    DenseOutput = Dense(1, activation='sigmoid')(RNNOutput)
end for*Output Layer*:
output = Dense(1, activation='sigmoid')(DenseOutput)
Output:
return output =0

V. Results

Here we will investigate the impact of different model architectures on deep learning-based sequence modeling tasks. Specifically, we analyze the effect of varying the number of layers in convolutional neural networks (CNNs) and recurrent neural networks (RNNs), as well as the use of bidirectional Long Short-Term Memory (LSTM) and different types of RNN cells. Our findings demonstrate the relationship between the number of trainable parameters, model performance, and convergence speed.

Designing an optimal model architecture for such tasks requires careful consideration of various factors, including the number of layers, type of cells, and parameter count. In this paper, we delve into these considerations and present our findings. We implemented several variations of deep learning architectures, each with different configurations. We trained these models using standard optimization techniques, monitoring key performance metrics and training progress.

We observed that increasing the number of CNN layers resulted in a reduction in the number of trainable parameters. This phenomenon can be attributed to the network's ability to learn higher-level features as it goes deeper. Despite the parameter reduction, the performance remained comparable or improved, suggesting that deeper CNN architectures can capture more complex patterns efficiently. Conversely, increasing the number of RNN layers resulted in an increased number of trainable parameters. Each additional layer introduces more weights and biases, expanding the model's capacity to capture intricate dependencies in the input sequence. However, this also requires a larger computational overhead during training.

We examined the impact of replacing LSTM with bidirectional LSTM, a variant that processes input sequences in both forward and backward directions. While bidirectional LSTM increased the number of trainable parameters, we found no significant difference in the results compared to regular LSTM. However, it did reduce the number of training epochs required for convergence, indicating its potential for faster training. Our analysis showed that both LSTM and Gated Recurrent Unit (GRU) architectures yielded similar results. These two types of RNN cells have comparable capabilities for modeling sequential data. Researchers and practitioners can choose either based on considerations such as computational resources or specific task requirements.

The best results were achieved with a combination of 3 CNN layers and 2 GRU layers. This configuration demonstrated superior performance with reduced trainable parameters (139,361). The model achieved a loss of 0.334, an accuracy of 0.876, precision of 0.810, and an AUC score of 0.876 after 31 epochs of training. The comparison of Results are shown in detail in Table 1.

VI. Conclusion

In conclusion, this research paper investigated the effectiveness of using a convolutional neural network (CNN) and recurrent neural network (RNN) model for character-level phishing email and SMS spam detection. Our study found that the proposed CNN-RNN model, which takes character-wise one-hot encoded vectors as input, can be highly effective in mitigating the risks associated with these types of cyberattacks.

The results of our experiments revealed that our proposed CNN-RNN model achieved an accuracy of 0.8759 and an AUC score of 0.88 for phishing email detection. For SMS spam detection, our approach achieved an accuracy of 0.9854 and an AUC score of 0.9962. These results demonstrate the potential of the CNN-RNN model to accurately detect and prevent phishing email and SMS spam attacks.

Overall, our research highlights the critical role that deep learning models, such as the CNN-RNN, can play in mitigating the risks associated with cyberattacks. By leveraging the power of deep learning to detect and prevent phishing email and SMS spam attacks at the character level, individuals and businesses can take proactive measures to protect against financial losses, privacy violations, and identity theft.

While our study focused on the effectiveness of the CNN-RNN model for character-level phishing email and SMS spam detection, future research could explore other types of deep learning models and evaluate the effectiveness of these models in detecting and preventing various types of cyberattacks. Additionally, further investigation could be conducted into the development of more robust and accurate deep learning models for detecting and preventing phishing email and SMS spam attacks at the character level.

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,

[16] Y. Li, Z. Yang, X. Chen, H. Yuan, and W. Liu, "A stacking model using URL and HTML features for phishing webpage detection," Future Generation Computer Systems, vol. 94, pp. 27-39, May 2019. DOI: 10.1016/j.future.2018.11.013

[17] A. Mishra and B.B. Gupta, "Intelligent phishing detection system using similarity matching algorithms," International Journal of Information and Communication Technology, vol. 12, no. 1/2, pp. 51, January 2018. DOI: 10.1504/IJICT.2018.089022

[18] A. Correa Bahnsen, E. Contreras Bohorquez, S. Villegas, and J. S. Vargas, "Classifying phishing URLs using recurrent neural networks," in Proceedings of the 2017 APWG Symposium on Electronic Crime Research (eCrime), April 2017. DOI: 10.1109/ECRIME.2017.7945048

[19] Y. Kim, "Convolutional Neural Networks for Sentence Classification," arXiv:1408.5882 [cs.CL], September 2014. Available: https://arxiv.org/abs/1408.5882

[20] X. Zhang, J. Zhao, and Y. LeCun, "Character-level Convolutional Networks for Text Classification," arXiv:1509.01626 [cs.LG], April 2016. Available: https://arxiv.org/abs/1509.01626