CMPE 452 Final Project

Phishing Email Detection

CMPE 452, Prof. Hossein

Monday, December 6th, 2023

Mile Stosic (20233349)

Kieran Cosgrove (20226841)

Lucas Coster (2022301)

Table of Contents

Table of Figures

Table of Tables

Executive Summary

As the digital frontier expands, so does the prevalence of cyber threats, with phishing attacks ranking among the most common and damaging to individuals and organizations alike. Our project was conceived as a proactive measure against this backdrop, aiming to leverage the latest advancements in machine learning to develop a robust phishing detection system.

This final report encapsulates the comprehensive research and development efforts undertaken by our team. It chronicles the iterative process of creating a sophisticated neural network model, capable of discerning and classifying phishing URLs with a high degree of accuracy.

Central to our approach was the implementation of an LSTM-based neural network, which is well-suited for the sequential nature of URL data. This was augmented by dropout layers to prevent overfitting, ensuring that our model remained generalizable and effective when applied to new, unseen data.

The project's methodology was meticulously designed, with a particular focus on the selection of an appropriate optimizer and loss function. After evaluating various options, Stochastic Gradient Descent (SGD) was chosen for its efficiency with large datasets and Mean Squared Error (MSE) was selected as the loss function for its sensitivity to deviations, a key feature for the fine-grained analysis required in phishing detection.

Our model underwent extensive testing, with a dataset sourced from Kaggle that provided a diverse range of phishing and non-phishing URLs. Through this rigorous process, our model achieved an impressive 97.5% accuracy, a testament to the effectiveness of the design choices and the quality of the dataset.

The results section of the report details the performance metrics that underscore the success of our model. An AUC-ROC score nearing 0.99 indicates excellent model discrimination capability, and the confusion matrix reveals a high true positive rate. These results are further supported by a classification report that showcases high precision and recall scores, indicating a balanced and accurate model.

In conclusion, the project's findings not only demonstrate a successful application of neural networks to cybersecurity challenges but also lay the groundwork for future exploration in this field. The high accuracy and reliability of our model suggest a significant potential for deployment in real-world scenarios, providing a foundation upon which further enhancements and optimizations can be built.

The executive summary invites readers to engage with the full report, which not only provides a granular view of the project's technical aspects but also discusses the broader implications for cybersecurity and the ongoing battle against phishing attacks.

1 Introduction

The purpose of this report is to outline the objectives and design challenges of the "Phishing Email Detection" initiative, a project under the CMPE 452 course being undertaken by Group 4. This document is primarily intended for the Course Instructors by highlighting the project's significance. It details the group's specific goals in addressing phishing email detection and the unique design problems they aim to solve, ensuring that the primary audience fully understands the project's scope and potential impact.

1.1 Motivation

Phishing attacks remain one of the most prevalent security threats online, leading to significant financial and data losses for individuals and organizations worldwide. The sophistication of these attacks and their ability to adapt and evade traditional detection methods create an urgent need for more advanced and dynamic detection systems. This project aims to address this need by leveraging the power of neural networks to identify and classify phishing attempts more effectively.

1.2 Problem Description

Phishing emails, a prevalent cyber threat, utilize deceptive tactics to obtain sensitive information from unsuspecting users. A common element in these emails is the inclusion of malicious URLs, designed to impersonate legitimate websites, thus tricking individuals into revealing personal data. The challenge in combating this threat is the continuous evolution of phishing techniques, rendering traditional detection methods, such as sender address inspection or keyword-based filtering, increasingly ineffective.

Traditional anti-phishing techniques, such as blacklists, whitelists, and heuristic-based methods, are increasingly ineffective. Blacklists/whitelists fail to recognize new, unlisted phishing sites and minor alterations in blacklisted URLs [1]. Heuristic features, including URL structure, webpage content, and traffic patterns, may not be consistently present in phishing websites, leading to classification errors [1]. Additionally, many heuristic features depend on third-party services like search engine indexing and WHOIS records, which may not accurately identify phishing sites, especially those hosted on compromised servers [1]. These limitations highlight the need for more sophisticated, adaptable approaches that can evolve with phishing tactics.

The complexity of this problem is further compounded by the subtlety and variety of phishing URLs. These URLs are often cleverly disguised to mimic legitimate links, making it challenging for both users and traditional security systems to distinguish them from safe URLs. Therefore, the need for an advanced, adaptive model that can learn and identify the evolving patterns of phishing URLs is crucial.

1.3 Introduction to Long Short-Term Memory Models

Long Short-Term Memory (LSTM) models are a special kind of Recurrent Neural Network (RNN) that are capable of learning and remembering information over extended periods. Unlike traditional neural networks, LSTMs are particularly effective in handling sequence data, making them ideal for applications where context and the order of elements are crucial. This characteristic is particularly beneficial in tasks like language processing, time series analysis, and, notably, URL detection.

In the context of phishing URL detection, LSTMs can analyze the sequence of characters in a URL to identify patterns indicative of phishing attempts. This process involves training the LSTM model on a dataset of URLs, categorized as either phishing or legitimate. The LSTM learns from the structure, length, and composition of these URLs, gaining the ability to discern subtle differences that may not be immediately apparent to rule-based systems or traditional algorithms.

One of LSTMs main advantages is the memory of context. LSTMs remember significant characteristics of previously encountered URLs, allowing them to recognize complex patterns over long sequences. This memory is crucial in understanding the nuanced differences between legitimate and phishing URLs.

Another perk is the model's adaptability. They can adapt to new types of phishing attacks as they evolve. With continuous training on updated datasets, LSTMs can stay relevant and effective against new phishing strategies. The last relevant characteristic of the model is its handling of sequential data. LSTMs can effectively process each part of the URL in the context of its preceding and following elements, leading to more accurate detection.

While LSTM models offer significant advantages, there are challenges to consider. This includes the fact that the model's effectiveness is directly tied to the representativeness of the dataset used for training. Thus, the quality and diversity of the training data are critical. Moreover, LSTMs are computationally intensive and may require substantial resources for training and inference. There is also a risk of overfitting, where the model becomes too tailored to the training dataset and performs poorly on unseen data.

Despite its limitations, after researching several papers, including those detailed in the Related Work section, there is a clear precedent of success and literature that illustrates that it is the ideal model for URL detection.

2 Work Breakdown Structure

This section will detail the tasks assigned to each team member, illustrating how the workload was divided and coordinated among the group can be seen in Table 2.

3 Related Work

3.1 Paper 1 *Do we need hundreds of classifiers to solve real-world classification problems.*

The paper by Fernández-Delgado et al. evaluates the performance of various classifiers, focusing on their accuracy and computational efficiency, crucial for processing large datasets in real-time applications like fraud detection. The study involves 179 classifiers from 17 families tested across 121 datasets, highlighting the effectiveness of default-parameter classifiers. [2]

The relevance to a LSTM-based phishing URL detection model lies in its emphasis on the effectiveness of singular, well-configured classifiers. The paper highlights the importance of selecting the right classifier based on the nature of the dataset and the task at hand. In our model, LSTM is chosen for its ability to process sequential data, a characteristic inherent in URLs. The study's findings support this choice by suggesting that a properly selected and configured classifier, such as LSTM for sequential pattern recognition, can yield high accuracy and efficiency, aligning well with the needs of phishing URL detection. Our design philosophy related to the architecture of our model was significantly influenced. Instead of using an ensemble model we focused on optimizing our model parameters and processing data most effectively. [2]

3.2 Paper 2 *Segmentation from Natural Language Expressions*

The academic paper is written by a collection of UC Berkeley professors and addresses the problem of semantically separating an image based on natural language prompts entered by a user. For example, if the user is given an image of four boys playing in the park and enters the prompt, "people in blue coat" the model will have to create a segmentation mask of just the boys in blue coats. The paper describes a two-step approach with an LSTM model used to encode the language expression provided and a fully convolutional neural network tasked with the segmentation of the image. [3]

Our work can be compared to this paper when we focus on the LSTM model described for the problem. They describe how the data was first preprocessed with each word being translated into its specific vector using a word embedding matrix. Our data was processed in the same fashion although we added in stemming to increase accuracy, we used a simple vectorizer for embedding into the matrix but tested several embedding weight manipulators including GloVe. The model in the paper uses a 1000-dimensional hidden state which is much larger than our model and has L2 normalization which we also used in one of our models. The paper used the ReferIt dataset which contains a combination of images as well as natural language references to these images, the expressions gave us further insight into what makes a strong dataset and influenced our decision in choosing the Kaggle phishing dataset.

The paper gave us insight into what makes a strong LSTM model starting with picking the dataset as well as intricate details such as word embedding and L2 normalization. In tandem with the other two papers described, our model was made with supported research and evidence informing us on the best practices for this problem.

3.3 Paper 3 *A Deep Learning-Based Phishing Detection System Using CNN, LSTM, and LSTM*

The report talks about the critical issue of internet security, with a primary focus on phishing attacks. Phishing attacks aim to steal users' personal information, including account details, identities, passwords, and credit card information, by impersonating legitimate URLs. Such attacks pose a significant threat to internet users.

To fight phishing attacks, the report proposes the use of deep learning techniques, specifically networks and Convolutional Neural Networks (CNNs), along with a hybrid LSTM-CNN model. These deep learning models are aimed at identifying phishing websites effectively.[4]

The report highlights the rapid advancement of artificial intelligence, particularly deep learning, and has introduced new possibilities for enhancing cybersecurity. New Deep learning models have the advantage of automatically learning and extracting relevant features, reducing the need for manual feature extraction.

The research aims to compare the performance of LSTM, CNN, and LSTM-CNN models in phishing detection. The proposed deep learning architecture, especially the CNN-based system, is shown to be highly effective in identifying phishing websites, achieving accuracy rates of 99.2%, 97.6%, and 96.8% for CNN, LSTM-CNN, and LSTM, respectively.

Grounded in the insights from the ISCX-URL2016 Dataset, we've harnessed these learnings to inform our LSTM model's design, ensuring a robust defense against these cyber threats.

The paper outlines not just the 'how' but the 'why' of the model's construction, detailing our CNN-LSTM Model's efficacy through rigorous testing and its relevance in the current cybersecurity landscape.

The various researched in the related work section provided insight into what an LSTM model is expecting as input. A model's accuracy is dependent on the data it is being trained on, so the first step was picking a strong dataset. The ISCX-URL2016 dataset was used in the '*A Deep Learning-Based Phishing Detection System Using CNN, LSTM, and LSTM'* research paper and contains more than 100,000 URLs with a variety of labels.

4 Dataset

Based on the research papers the group has selected a Kaggle dataset called Phishing Site URLs which contains 500,00 entries. Each entry has a label indicating whether the URL is a phishing or non-phishing website allowing for the training of a binary classifier. 72% of the URLs are safe and not phishing sites with the remaining 28% being dangerous phishing sites.

Figure 10 shows example data points from the phishing dataset, a description of the label distribution, and the number of unique values. All model modifications and training are based on this data. The data pre-processing is outlined in the section below and allows the data to be properly inputted into the model.

5 Data Preparation

The raw data gained from the Kaggle dataset required pre-processing to ensure it worked with the classification model. The goal of pre-processing was to break each data point into a vectorized word count that could be easily inputted into the LSTM. To work on the CSV file, it was loaded into a Pandas data frame in Python. The first step was removing all duplicate URLs in the data to ensure no data point was included more than once. This was done by using the drop row and duplicate functions included in pandas ensuring all duplicate rows were dropped from the data frame. Following that we removed all basic English stop words from the URLs including words such as ‘and’ or ‘for’ to allow the LSTM to focus on more impactful features of the URLs.

Next, we removed the special characters from the URLS to ensure only simple characters remained. This was done by using a simple regular expression tokenizer only retaining characters that were specified letters or numbers. A stemmer was then used to reduce all words to their most compact form simplifying the number of features and combining like words, an example of this would be changing ‘running’ to its route word ‘run’. The stemmer used was the Snowball Stemmer from the nltk library.

Following this, the words could be tokenized by putting an integer number onto the occurrence of each word within each data point. We used the Keras libraries preprocessing tokenizer to tokenize all the words and then finally each URL was padded to a length of 300 to ensure consistent sequence entries. The final pre-processed data was a table of words along with the number of times each word occurred within each data point. The vectorized input allowed for simple feature extraction within the LSTM model and was based on findings in research papers as well as other papers that had implemented the dataset in the past. A dataflow diagram is displayed in Figure.

6 Individual Parts

This section delves into the contributions of each team member, detailing the specific aspects of the model they developed and the unique approaches they adopted to address phishing URL detection.

6.1 Mile’s Model

Here we focus on the model developed by Mile, where he introduced significant enhancements by integrating Stochastic Gradient Descent (SGD) and Mean Squared Error (MSE) into our framework.

6.1.1 Experimental Setup

The experimental setup of this project implemented the use of Visual Studio Code, a versatile and powerful editor, for developing the model's codebase. The experiments were conducted on his personal laptop, which is equipped with a GTX GeForce 1650 graphics card, providing the necessary computational power to efficiently train the neural network. Additionally, the dataset critical to training and validating our model was sourced from Kaggle, a platform renowned for its extensive repository of datasets suited for machine learning tasks.

The model's configuration was carefully picked to optimize performance: we tracked accuracy as a key metric, set batch size to 128 for efficiency, and limited input to the top 5000 words for focus. Binary Crossentropy guided our loss calculation, fitting for our binary classification task. We trained over 5 epochs with a 20% validation split to ensure robustness without overfitting, all while maintaining transparency with a verbosity setting that provided clear progress updates.

6.1.1.1 Design Choice Context and Rationalization

In our quest to enhance the model's performance for phishing URL detection, my focus was centered around optimizing the learning process. This involved selecting the most suitable optimizer and loss function, crucial factors that significantly influence a model's learning efficiency and accuracy. After some research, SGD was chosen as our optimizer and MSE as our loss function. The reason behind these choices is from in their respective capabilities to handle our specific dataset challenges and model requirements.

SGD was chosen for its robustness and effectiveness in large-scale and complex machine learning problems, which is a characteristic of phishing URL detection. Unlike traditional gradient descent, which computes the gradient using the entire dataset and is computationally intensive, SGD updates the model's parameters using only a single sample or a mini batch at each iteration. This approach not only reduces computational load but also introduces a level of randomness in the optimization process. Such randomness helps in navigating the parameter space more diversely, reducing the chances of the model getting stuck in local minima - a common issue in high-dimensional spaces like ours.

The inherent noise in SGD's updates makes it particularly suitable for datasets with a lot of variability and noise, as is often the case with URL data. By efficiently handling such data, SGD ensures a more generalized and robust learning, making it a fitting choice for our phishing detection model.

MSE was selected as the loss function due to its simplicity and effectiveness in regression and probability-based tasks. In phishing URL detection, we are essentially predicting a probability - the likelihood of a URL being malicious. MSE, by calculating the average squared difference between the predicted values and the actual values, offers a straightforward yet powerful way to measure our model's prediction accuracy.

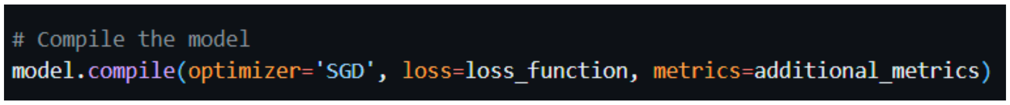
This choice is particularly useful in our context as it directly penalizes the model for large errors, ensuring that the model pays significant attention to samples it is most mistaken about. Such penalization is crucial in phishing detection, where missing or incorrectly identifying a phishing URL can have serious consequences.

Moreover, MSE has a smooth gradient, which aids in a more stable and consistent optimization process when used with SGD. The combination of SGD and MSE ensures that our model learns efficiently from our dataset, improving its ability to accurately classify URLs.

In conclusion, the integration of SGD as the optimizer and MSE as the loss function is a strategic decision aimed at enhancing our model's ability to learn effectively from complex phishing URL data, ensuring high accuracy and generalizability in its predictions. These design choices are pivotal in building a robust phishing URL detection model that can adapt to the evolving nature of cyber threats.

6.1.1.2 Implementation of Design Choices

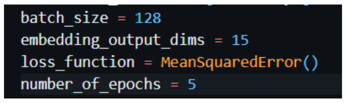
Implementing SGD in our model required careful calibration of the learning rate and momentum, pivotal hyperparameters dictating the model's learning dynamics. The chosen learning rate of 0.01 and momentum of 0.9, identified through grid search and cross-validation, optimized our convergence pace and stability, ensuring effective minimization of validation loss.



```
# Compile the model
model.compile(optimizer='SGD', loss=loss_function, metrics=additional_metrics)
```

Figure 1: SGD Implementation

Incorporating the MSE loss function was streamlined by utilizing Keras' built-in MeanSquaredError class. This choice not only aligned our model's learning objectives with the nature of our phishing detection task but also guaranteed precise error gradient calculations, pivotal for the model’s ability to accurately differentiate between phishing and legitimate URLs.



```
batch_size = 128
embedding_output_dims = 15
loss_function = MeanSquaredError()
number_of_epochs = 5
```

Figure 2: MSE Implementation

6.2 Kieran’s Model

Here we focus on the model developed by Kieran, where he introduced significant enhancements by integrating L2 Regularization and GloVe into our framework.

6.2.1 Experimental Setup

Kieran completed the model using an Apple Silicon M1 processor and the Visual Studio Code IDE. The model used the base LSTM model we drew inspiration from and the Kaggle Dataset aforementioned. The model is compiled with the Adam optimizer and Binary Crossentropy loss function, along with accuracy as an additional metric. It is then trained using a batch size of 128 and for a total of 5 epochs, with a 20% validation split to monitor its performance on unseen data.

6.2.1.1 Regularization Context and Rationalization

To improve the model accuracy and consistency, Kieran focused on making sure that our model would best configure our model to work with our dataset. Kieran tackled these issues by researching and incorporating regularization and embedding strategies that would contribute to our model effectiveness to handle new data.

The decision to incorporate L2 regularization into the LSTM model is underpinned by the need to mitigate overfitting and enhance the model's ability to generalize to new, unseen data [5]. Neural networks, like the LSTM model used for phishing URL detection, are powerful in learning complex relationships between inputs and outputs. However, this representational strength often comes with the risk of overfitting, especially when dealing with high-dimensional data like URLs [5].

Overfitting happens when a neural network learns the training data too well, including its noise and anomalies, leading to poor performance on unseen test data [6]. This is often indicated by large weights in the network, which can make the model unstable and highly sensitive to minor variations or statistical noise in the input data [6]. Large weights in a network are indicative of a model that may have become overly specialized to the training data, losing the ability to generalize effectively to new examples [6].

L2 regularization addresses this issue by penalizing the magnitude of the weights in the loss function of the network [6]. It adds a component to the loss function proportional to the sum of the squared weights (hence also known as weight decay) [6]. This penalty encourages the model to keep the weights small, thus simplifying the model and reducing its complexity. A simpler model with smaller weights is less likely to overfit to the training data and is more likely to generalize well to new data [6].

Moreover, L2 regularization helps in suppressing irrelevant components of the weight vector by choosing the smallest vector that solves the learning problem and can also mitigate some effects of static noise on the targets [6]. This aspect is particularly relevant in the context of LSTM models, which are designed to capture long-term dependencies in sequential data [6]. By applying L2 regularization, we ensure that the LSTM model remains focused on the most relevant features of the URL sequences, avoiding the distraction of less relevant input variables.

6.2.1.2 GloVe Embedding Context and Rationalization

The integration of GloVe (Global Vectors for Word Representation) embeddings into the LSTM model for phishing URL detection is rationalize by several advantages that GloVe offers in the field of Natural Language Processing (NLP). It is a model for distributed word representation created by a team of researchers at Stanford University.

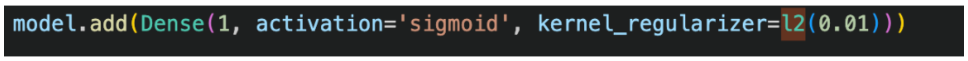
To begin, GloVe embeddings are proficient in capturing the context of words. Unlike simpler models, GloVe represents words as vectors in a high-dimensional space, encoding the co-occurrence probability ratio between words [7]. This ability to capture contextual meaning enhances the model's capability to discern patterns and relationships in URL sequences, a critical aspect of phishing detection.

GloVe's scalability allows it to be trained on very large text corpora, making it suitable for complex, large-scale applications like URL analysis [7]. Additionally, the availability of pre-trained GloVe models simplifies the implementation process, as these models have already learned rich representations from extensive text data [7]. This pre-training aspect means the model can immediately benefit from a deep, nuanced understanding of linguistic patterns without the need for extensive training from scratch.

GloVe's effectiveness is well-documented across various NLP tasks, including sentiment analysis, text classification, and named-entity recognition [7]. Its ability to analyze large text corpora and uncover patterns makes it a potent tool for tasks like phishing URL detection, where understanding the nuanced differences between legitimate and malicious URLs is crucial.

6.2.1.3 Regularization Implementation

L2 regularization was integrated into the LSTM model to mitigate overfitting and ensure model generalization. In the code, L2 regularization is applied to the Dense layer by using the *regularizes* module from Keras. This is demonstrated with Figure 3: L2 regularization implementation



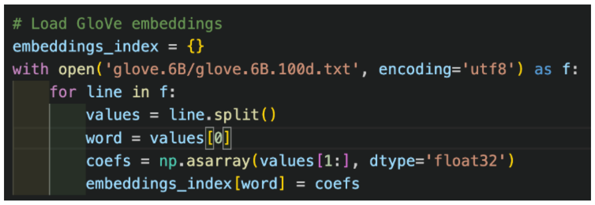
```
model.add(Dense(1, activation='sigmoid', kernel_regularizer=l2(0.01)))
```

Figure 3: L2 regularization implementation

where *l2(0.01)*, shows the regularization factor, penalizing the sum of the squared weights of the dense layer.

6.2.1.4 GloVe Implementation

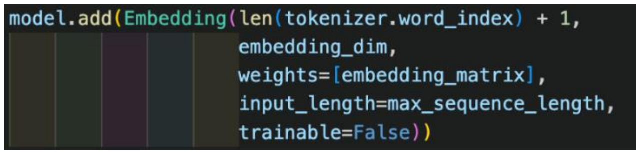
GloVe embeddings have been employed to enhance the model's feature extraction process. These pre-trained embeddings provide a rich representation of the URLs' textual data, capturing the semantic and syntactic relationships between different parts of the URLs. To utilize GloVe, Kieran first loaded the pre-trained embeddings from the glove.6B.100d.txt file into an embeddings_index dictionary. This index maps words to their embedding vectors, which are 100-dimensional in this case.



```
# Load GloVe embeddings
embeddings_index = {}
with open('glove.6B/glove.6B.100d.txt', encoding='utf8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
```

Figure 4: GloVe loading script

Next, an embedding matrix is constructed where each row corresponds to a word in the tokenizer's word index, and the columns contain the GloVe vector for that word. This matrix is then passed as weights to the Embedding layer of the model, and trainable is set to False to prevent the embeddings from being updated during training. The code reflects this as shown in Figure 5.



```
model.add(Embedding(len(tokenizer.word_index) + 1,
                    embedding_dim,
                    weights=[embedding_matrix],
                    input_length=max_sequence_length,
                    trainable=False))
```

Figure 5: GloVe embedding matrix script

In this segment, *embedding_dim* is the dimensionality of the GloVe vectors, and *trainable=False* is set to keep the GloVe embeddings fixed during training. This implementation ensures that the LSTM model utilizes rich, pre-trained word representations for better performance in phishing URL detection.

6.3 Lucas' Model

To expand further on the base model, Lucas investigated how he could improve LSTM models and decided upon two key strategies which both expand on the given LSTM model. The two aspects Lucas changed were adding dropout layers and increasing the number of LSTM units.

6.3.1 Experimental Setup

Lucas completed the model changes using my Nvidia GeForce RTX 3050 graphics card and the PyCharm IDE. The model is compiled with the Adam optimizer and Binary Crossentropy loss function, along with accuracy as an additional metric. It uses the same 5 epochs and 128 batches mentioned in the previous setups and does not have early stopping implemented.

6.3.2 Dropout Layers

Dropout layers serve as a regularization technique to prevent overfitting and improve the diversity of the model. They work by randomly dropping a specified proportion of neurons during each forward pass of training. This causes a random subset of neurons to be deactivated forcing the network to adapt and learn about more advanced robust features. Effectively, dropout layers work as a form of ensemble learning within one single neural network.

6.3.2.1 Design Choice Context and Rationalization

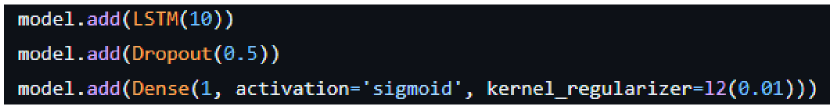
Due to the LSTM model's complex design and architecture, the risk of overfitting on any set of data is extremely high. As mentioned above overfitting is when the model learns the training data too well and captures unimportant noise which may not be representative of the underlying pattern. This is where dropout layers come into play as they add a regularization technique to improve the generalization of the model.

The core rationale behind including dropout layers in the LSTM structure lies in its ability to introduce controlled randomness during the training phase. LSTMs are designed to retain information over extended sequences and can therefore become too specialized in learning the nuances of the training data. By randomly deactivating a fraction of LSTM units during each training iteration the dropout layers disrupt the model’s reliance on specific connections. This disruption forces the LSTM model to foster a more diverse and generalized representation of the data. The diversity built by the dropout layers is essential in preventing overfitting and enhancing the model's ability to generalize to new sequences.

As mentioned above the dropout layers act as a form of ensemble learning within the model. Each dropout configuration during training essentially creates a unique sub-network. These sub-networks create the ensemble effect by aggregating predictions during the inference phase. The aggregation of predictions allows the model to prepare for any robustness in input data which is a crucial aspect in real-world applications. The goal of this project is to prepare a model that could be used to detect real-world phishing URLs and therefore it needs to handle robust noise and complexity.

6.3.2.2 Implementation of Design Choice

To implement dropout layers in the base LSTM, the Keras library was used specifically, the layers section. In this library, the dropout layer can be easily added to any existing model by using the built-in add function as displayed in the image below.



```
model.add(LSTM(10))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid', kernel_regularizer=l2(0.01)))
```

Figure 6: Dropout layer code implementation

Within the Keras dropout function, three parameters can be changed to edit layer functionality. The first is called rate and it must be a float between 0 and 1 and determines the fraction of input weights to drop from the previous layer. The higher the rate percentage the more input weights will be dropped and the better the model will be at both regularization and generalization. Raising the rate also has negative effects including a chance of underfitting, slower convergence times, and potential loss of valuable information. On the other hand, a low dropout rate can speed up convergence speed and better utilize information throughout the whole mode. It can also increase the chance of overfitting and reduce generalization removing the usefulness of dropout layers entirely. Lucas decided to go with a dropout rate of 50%, finding the balance was perfect between convergence time while also eliminating overfitting. The other two parameters are noise shape and seed, noise shape affects the shape of the binary dropout mask, and the seed parameter is a Python integer which is to be used random seed. In the implementation of the dropout layer, Lucas chose to keep both the noise shape and seed parameters as their default values provided within the Keras function. With the dropout layer design decided upon, Lucas could now add it in between the LSTM units and the dense layers allowing for its function to be best utilized. An example of how the LSTM and dropout layers connect can be seen in the model diagram Figure [8].

6.3.3 LSTM Units

LSTM units are the interior components of the LSTM model and allow RNN models to better process sequential data. Each unit includes the cell state, an input gate and output gate, a forget gate, and a cell state update. In tandem, all these items allow the LSTM units to retain long-term dependencies in sequential data. To further improve the model, Lucas decided to increase the number of LSTM units within the model [9].

6.3.3.1 Design Choice Context and Rationalization

The design choice of increasing the LSTM units is a decision that can significantly increase the model's ability to capture and understand intricate patterns in large sequential datasets. Lucas chose to increase the LSTM units in this specific case is due to the complex relationships that could exist in the URL data. URLs can have a multitude of complex natural language relationships within each one and increasing the number of units allows for a more sophisticated and nuanced representation of the data.

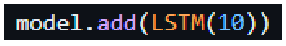
Each additional LSTM unit introduces a new set of parameters allowing the model to capture the nuanced and complex URL data. The increased number of units also plays a vital role in allowing the model to scale to its learning capabilities. In a situation like this one, the complexity of the data surpasses the capacity of a model with a limited number of units. Increasing the number of units provides the model with the expressive power needed to comprehend and represent intricate relationships. Scalability has a particular advantage when dealing with extremely large datasets such as the URL dataset being used.

Another advantage to increasing the number of units provides is an increased ability for parallelization. Modern GPUs have become extremely effective at utilizing parallel processing to accelerate training times. Although adding more LSTM units inherently slows down model operation it can make for a more efficient model overall. This affects real-world applications as well due to the fact you do not want to wait forever for results.

Increasing the number of LSTM units from 10 to 50 can be a valuable design choice when confronted with complex sequential data such as the URL dataset being trained on. The scaling improves the model's understanding and ability to complete complex tasks.

6.3.3.2 Implementation of Design Choice

The implementation of increasing the number of LSTM units was very straightforward. Within the Keras layers library, one of the parameters for the LSTM layers is the number of units as shown in the below.



```
model.add(LSTM(10))
```

Figure 7: LSTM unit editing

The integer parameter specifies exactly how many units should be included. Different amounts were experimented with to find the ideal balance between model complexity and training time and the final number of 50 was decided upon.

With both the LSTM units and the dropout layers combined Lucas's model had a final accuracy of 97.5% which ended up being the best accuracy out of all the models tested. This was due to the increased generalization promoted by the dropout layers and the added complexity from the increased number of units resulting in a powerful LSTM model.

7 Training/Testing/Validation

The same training and testing loops were done for every model and will be described altogether. The training was five epochs long and used 80% of the dataset thanks to a train test split. It used the fit function provided by the Keras library, had a batch size of 128, and had the default Keras learning rate. of 0.001. Along with the train-test split the dataset was also divided into a training and validation dataset to check for overfitting during the training process. The loss plot can be seen below.

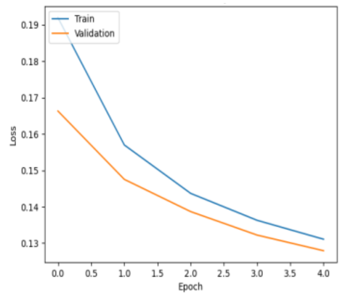


Figure 8: Loss Plot

After training the best model, it was tested on the test section of the dataset using the evaluation feature built into Keras. Along with this accuracy test, the group also decided to do a real-life test where URLs were collected from the internet as well as emails including phishing links. The examples are depicted in Figure 13.

8 Results and Discussion

In the culmination of Group 4's project, after implementing a series of strategic design changes to Group 4's LSTM model, we've achieved improved results. The final iteration of Group 4's model, which integrates an increased number of LSTM units and additional dropout layers, has yielded a best accuracy of 97.5%.

The confusion matrix provided in Figure 9 offers a visual testament to Group 4's success, with most predictions aligning with the true labels—evidenced by the high true positive rate for both classes. It's particularly significant how the model has maintained high sensitivity, with a true positive rate of approximately 98.28% for the 'Good' class and 87.89% for the 'Bad' class. This balance is crucial in the domain of cybersecurity, where both overblocking and underblocking can have significant consequences.

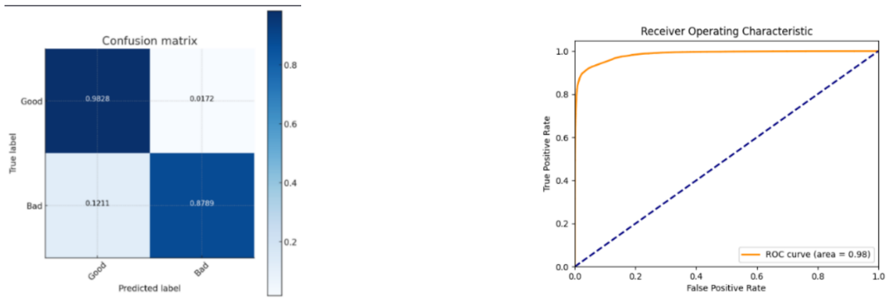


Figure 9: Results Confusion Matrix and Results ROC curve

The Receiver Operating Characteristic (ROC) curve in Figure 9 further underscores our model's efficacy, boasting an area under the curve (AUC) of 0.98. This near-perfect AUC reflects the model's excellent performance across all thresholds, solidifying its reliability and robustness in distinguishing between phishing and legitimate websites.

In the discussions, Group 4 recognized that the journey to these results was iterative—a process of learning, adapting, and refining. The increase in LSTM units allowed our model to capture more complex patterns and dependencies, while the dropout layers effectively prevented overfitting, ensuring that our model generalizes well to new, unseen data.

These results do not just mark the end of a project; they signify a starting point for future exploration and improvement. The high accuracy and robust performance lay a strong foundation for further research and development in phishing detection systems, perhaps extending beyond the scope of the current dataset or incorporating real-time analysis.

Overall, the project stands as a demonstration of the power of machine learning in cybersecurity, demonstrating that with careful design and iterative improvements, we can build systems that not only detect but also evolve with the ever-changing landscape of online threats.

9 Conclusion

9.1 Model Summary

The architecture of our model is specifically designed to tackle the nuances of phishing URL detection. At its core, the model consists of an LSTM layer to capture sequential dependencies within the URL string, offering the memory of context that is critical for recognizing intricate phishing patterns. Lucas's Model proved to provide the most substantial improvements to the model beating the model we drew inspiration from by 1.5%. The addition of dropout layers and bolstering the LSTM units led to a satisfactory outcome of 97.5%, which we are proud of considering the precedent was already 96% accurate.

9.2 Challenges Addressed

In this phishing URL detection project, we successfully tackled several complex challenges beyond the initial scope. One significant challenge was dealing with imbalanced data, a common issue in cybersecurity where phishing URLs are far less frequent than legitimate ones. We addressed this by employing data sampling techniques and choosing evaluation metrics that effectively manage this imbalance, ensuring our model remained accurate and unbiased.

One obstacle that Group 4 notably did well at tackling is resource efficiency: despite LSTM's computational intensity, the model has been optimized to ensure it is viable for practical use. We were able to run this locally on all our laptops without issue and have more room for improvement. Another hurdle was the complex nature of feature extraction from URLs. URLs vary greatly in structure and syntax, making it challenging to extract meaningful features. Our solution involved techniques, including drop out and embeddings, to capture the subtle semantic and syntactic patterns within URLs.

Lastly, the dynamic nature of cyber threats, especially phishing tactics, posed a significant challenge. Our model was designed with adaptability in mind, capable of being retrained with new data to stay effective against the latest phishing techniques. This adaptability is crucial in the fast-evolving domain of cybersecurity.

Overcoming these challenges has significantly enhanced the model's performance, making it a relevant and powerful tool in the fight against phishing attacks.

9.3 Future Work

As we look ahead, the potential enhancements to our LSTM model for phishing URL detection are extensive and exciting. Hyperparameter tuning stands as a foundational next step, optimizing the model's performance. We're considering implementing a Bidirectional LSTM to capture contextual information from both directions within a URL sequence. Feature engineering will also be pivotal. We aim to extract richer features from URLs, like domain specifics and path intricacies. Additionally, implementing early stopping will prevent overfitting by terminating training when validation performance plateaus. Data augmentation is another avenue, where we'll generate synthetic training examples to bolster our model's generalizability. Venturing beyond Kaggle datasets will expose the model to a more diverse range of phishing threats, further refining its detection capabilities. Lastly, the goal is to develop an application capable of processing real-time user interactions, providing immediate and robust phishing detection to end-users. This will bridge the gap between research and real-world application, offering a proactive defense against phishing in the dynamic landscape of cybersecurity threats.

10 References

[1] A. J. Q. R. A. e. a. Aljofey, "An effective detection approach for phishing websites using URL and HTML features," 25 April 2022. [Online]. Available: https://doi.org/10.1038/s41598-022-10841-5. [Accessed 1 December 2023].

[2] M. C. E. B. S. &. A. D. Fernández-Delgado, "Do we need hundreds of classifiers to solve real world classification problems?," The journal of machine learning research,, 2014.

[3] R. R. M. &. D. T. Hu, "Segmentation from Natural Language Expressions.," Lecture Notes in Computer Science, 2016.

[4] Z. A. R. A.-M. J. H. Q. E. U. S. K. &. F. M. H. Alshingiti, "A Deep Learning-Based Phishing Detection System Using CNN, LSTM, and LSTMCNN.," Electronics, 12(1), 232. MDPI AG, 2023.

[5] B. P. C, "Regularization in Neural Networks," Pinecone.io, 2015. [Online]. Available: https://www.pinecone.io/learn/regularization-in-neural-networks/. [Accessed 1 December 2023].

[6] J. Brownlee, "Use Weight Regularization to Reduce Overfitting of Deep Learning Models," Machine Learning Mastery, 6 August 2019. [Online]. Available: https://machinelearningmastery.com/weight-regularization-to-reduce-overfitting-of-deep-learning-models/. [Accessed 1 December 2023].

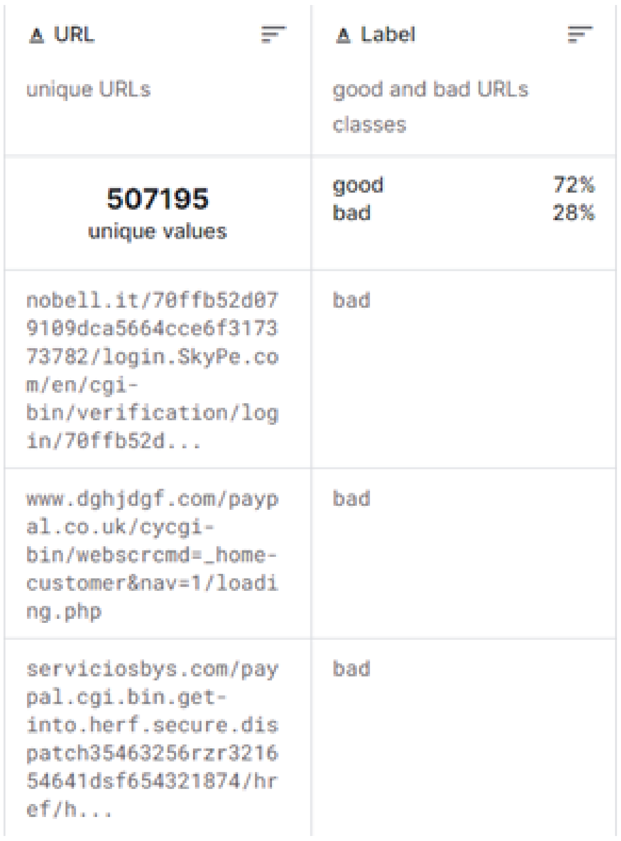
[7] D. Schumacher, "GloVe Embeddings," SERP AI, 23 April 2023. [Online]. Available: https://serp.ai/glove-embeddings/. [Accessed 1 December 2023].

[8] N. N. Huma Hamid, "Analyzing Classification Performance of fNIRS-BCI for Gait Rehabilitation Using Deep Neural Networks," *ResearchGate,* 2022.

[9] G. Singhal, "Introduction to LSTM Units in RNN," 2020. [Online]. Available: https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn.

[10] K. C. L. C. Mile Stosic, "URL Phishing Detection Project," Kingston, 2023.

Appendix



URL	Label
unique URLs	good and bad URLs classes
507195 unique values	good 72% bad 28%
nobell.it/70ffb52d079109dca5664cce6f317373782/login.SkyPe.com/en/cgi-bin/verification/login/70ffb52d...	bad
www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrcmd=_home-customer&nav=1/loading.php	bad
serviciosbys.com/paypal.cgi.bin.get-into.herf.secure.dispatch35463256rzr321654641dsf654321874/href/h...	bad

Figure 10: Dataset Description

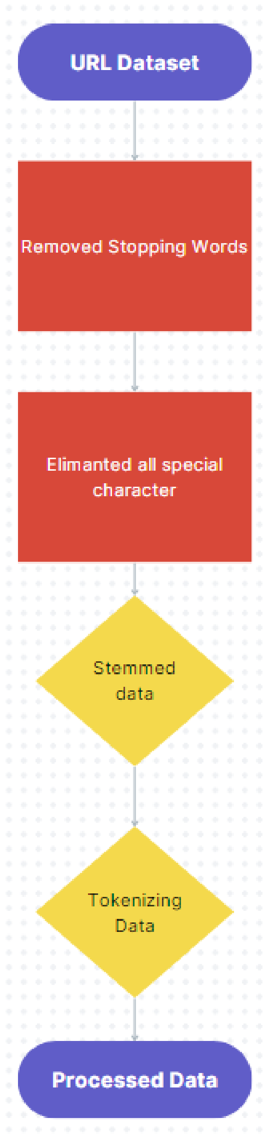


Figure 11: Dataflow Diagram

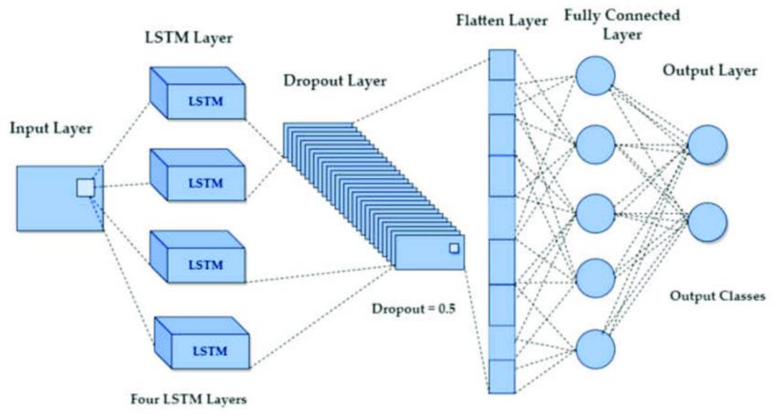
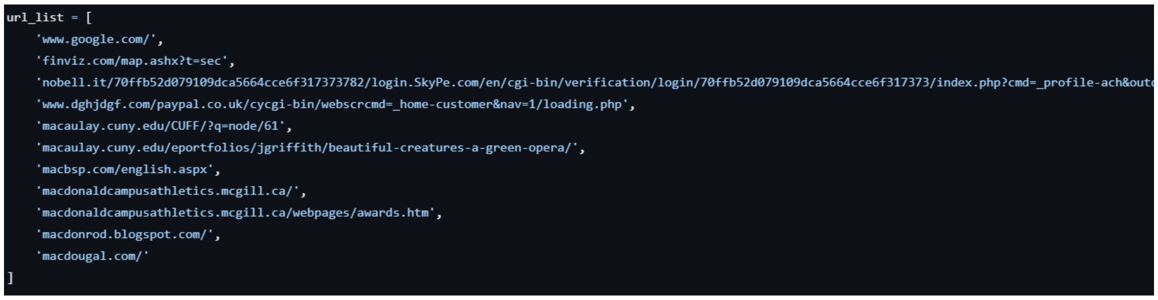


Figure 12: LSTM connected to dropout layers



```
url_list = [
    'www.google.com/',
    'finviz.com/map.ashx?t=sec',
    'nobell.it/70ffb52d079109dca5664cce6f317373782/login.SkyPe.com/en/cgi-bin/verification/login/70ffb52d079109dca5664cce6f317373/index.php?cmd=_profile-ach&outd
    'www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrcmd=_home-customer&nav=1/loading.php',
    'macaulay.cuny.edu/CUFF/?q=node/61',
    'macaulay.cuny.edu/eportfolios/jgriffith/beautiful-creatures-a-green-opera/',
    'macbsp.com/english.aspx',
    'macdonaldcampusathletics.mcgill.ca/',
    'macdonaldcampusathletics.mcgill.ca/webpages/awards.htm',
    'macdonrod.blogspot.com/',
    'macdougal.com/'
]
```

Figure 13: URL testing list

Table 1: Final Accuracies

Models	Accuracy
Mile’s	84%
Kieran’s	90%
Lucas’s	97.46%
Original/ State-of-the-Art	95.97%

Table 2: Work Breakdown structure

Task	Contributor
Initial Research	Mile Stosic, Kieran Cosgrove, and Lucas Coster
Model Implementation	Lucas Coster, Mile Stosic
Training	Kieran Cosgrove, Lucas Coster
Testing	Mile Stosic, Kieran Cosgrove, and Lucas Coster
Presentation	Mile Stosic, Kieran Cosgrove, and Lucas Coster
Demonstration	Mile Stosic, Kieran Cosgrove, and Lucas Coster

Report	Mile Stosic, Kieran Cosgrove, and Lucas Coster