Using Neural Networks to Identify Malicious URLs

Zach Dunn

1 Introduction

In today's digital age, one of the increasingly common challenges users face is determining whether a URL is safe to click. We constantly encounter links—whether in emails, on social media platforms, or through everyday web browsing—and each one presents a potential security risk. Malicious URLs can lead to phishing attacks, data theft, or unintentional downloads of harmful software. This project seeks to address that risk by leveraging machine learning techniques to identify and classify potentially dangerous URLs, thereby helping users avoid scams and protect their devices.

At the core of this project lies the application of artificial intelligence, particularly within the field of machine learning. A neural network model will be developed to handle the classification task—distinguishing between benign and malicious URLs based on extracted features. While traditional AI methods will not be employed for the preprocessing of raw URL data, the model will incorporate built-in text vectorization techniques to transform the various string-based components of each URL into a format suitable for learning. This integrated approach enables the system to process complex patterns in URL structures, improving its accuracy and effectiveness in detecting threats. After the development of this model, it will be compared to previous works to evaluate how well it performs.

2 Related Work

The paper *"Classification of Malicious URLs Using Machine Learning"* applied Decision Trees (DTs), Random Forests (RFs), Support Vector Machines (SVMs), and k-Nearest Neighbors (KNNs) to classify 650,000 URLs into benign, phishing, defacement, and malware categories. They used 16 extracted URL features and improved model training efficiency with instance selection methods (random selection, DRLSH, and BPLSH), alongside Bayesian Optimization for hyperparameter tuning. Random Forests consistently achieved the best balance of high precision, recall, and F1 scores (up to 92.18%) with fast training times, while SVMs were competitive but much slower [1]. Random instance selection usually led to better model performance than DRLSH or BPLSH. Feature analysis showed that the presence of "http" (vs. "https") was the most important indicator of malicious URLs. Across all models, phishing URLs were the hardest to detect accurately.

Next, the paper *"Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions"* presents a comprehensive review of 91 studies (2012–2021) applying machine learning and deep learning to malicious URL detection. Most methods use supervised learning with features extracted from URLs (lexical, content-based, and network-

based). Common algorithms included Random Forests (RF), Support Vector Machines (SVMs), Decision Trees (DTs), Naive Bayes (NB), and Logistic Regression (LR), while some recent studies also explored Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM), XGBoost, and ensemble models. RF and SVM were the most frequently used and achieved high accuracies (typically 95–99%), while a CNN achieved the highest reported accuracy of 99.98%. Lexical features were the most commonly used due to ease of extraction and processing speed, while Arabic-based studies largely lacked network feature usage. The review also highlights that balanced datasets, feature selection, and adaptation to evolving attack patterns (concept drift) are critical for maintaining detection performance.

Finally, the paper *"Detection of Malicious URLs Using Machine Learning"* compares classical machine learning models (Logistic Regression, Decision Trees, Support Vector Machines, and Neural Networks) and introduces Quantum Machine Learning (QML) using a Variational Quantum Classifier (VQC) for detecting fraudulent URLs. Using a publicly available dataset, classical models achieved high performance, with Decision Trees reaching 100% precision and 90% F1-score for fraudulent URL detection, and the third neural network variant achieving 96.35% accuracy. Support Vector Machines with RBF kernels also performed very well (95.86% AUC). Quantum models, though trained on a reduced dataset due to hardware limitations, achieved comparable test accuracy (up to 97%) but were slower and more sensitive to parameter choices. Overall, classical ML outperformed QML in full datasets, but QML showed promising results suggesting future potential in cybersecurity applications.

3 Methods

The primary goal of this project is to create a neural network to compare to the previous works mentioned above. The chosen neural network for this task was a convolutional neural network since this was shown to work well in previous studies. The steps taken to set up the network are as follows, first the URL data was acquired from Kaggle [4], then the data was preprocessed to extract the features that make up a URL. Finally, the model was built to accept the various features and transform them into numerical data which can then be fed into the other layers.

Starting with the data collection and processing, the dataset used for this project was the Malicious URLs Dataset from Kaggle [4] consisting of 651,191 URLs. The URLs were classified into four categories: benign, defacement, phishing, and malware. Before giving the data to the neural network, the URL features were extracted from the raw data. The features used include scheme, subdomain, second level domain, subdirectory, port, path, query, parameters, and fragment. For any feature that was not available for a particular URL an empty string was given instead. In addition to these features, a few derived features were used as well, including the character length of each URL, how many digits were in each URL, and how many special characters each URL had i.e. everything that was not numbers or letters.

To feed the features into the neural network, the first layers of the network consisted of eight input layers for the text-based features which included the subdomain, second level domain, top level domain, subdirectory, path, query, parameters, and fragment. An additional input layer was used for the remaining features which included the one-hot-encoded scheme, port, and the numerical data about the URLs. Before the text data could be passed into the network, however, it needed to be vectorized. For each text-based feature a new vectorizer was created and adapted to the data, each of which split the text at the character level. Additionally, the labels (benign, defacement, phishing, and malware) were converted to integers as well. Each of the text input layers were then passed into an embedding layer, a convolution layer, and a global max pooling layer. Furthermore, the numerical input layer was sent through a dense layer. The output from these layers were then concatenated and then passed into two sets of dense, batch normalization, and dropout layers. The final layer added was a dense output layer with 4 units for each of the categories and a softmax activation function.

This was implemented in Python [11] with the assistance of multiple packages. First, Pandas [5] was used to organize the data with its DataFrame structure. Numpy [6] was used to assist with array manipulation and mathematical operations. The internal Python library Urllib [11] was used to parse the URLs. Additionally, Tldextract [7] was used to further parse the URLs. TensorFlow [8] was used to create the CNN as well as vectorize the text. Sklearn [9] was used to one-hot encode, normalize data, encode labels, perform k-fold cross validation, and generate the classification reports and confusion matrices. Finally, MatPlotLib [10] and Seaborn [12] were used to display the reports.

Pseudocode:

split_url(url)
- Parse the URL into components (scheme, domain, subdomain, path, query, etc.) [7] [11]
- Extract additional elements like port and subdirectory manually in necessary
- Return all extracted parts as a dictionary

extract_features(text_series)
- For each string:
  - Compute length
  - Count digits
  - Count non-alphanumeric characters
- Return as a numerical feature array

create_text_vectorizer(max_tokens=500, output_seq_length=20)
- Create a character-level text vectorizer that tokenizes strings into fixed-length sequences [8]

prep_inputs(df)
- One-hot encode the scheme (e.g., http/https) [9]

- Get numeric features [6]
- Normalize all numerical features [9]
- Vectorize URL text fields [8]
- Encode class labels [9]
- Return combined numerical features, vectorized sequences, and labels

create_model(x_combined, sequence_data)
- Define input layers for: [8]
  - Combined numeric features
  - Each tokenized text sequence
- Use add embedding, convolution, and globalmaxpooling layers for text inputs [8]
- Combine all features and pass through dense layers [8]
- Output: softmax layer for multi-class classification [8]
- Compile and return the model

train_and_test(x_combined, sequence_data, y)
- Perform 5-fold cross-validation: [9]
  - For each fold:
    - Split data into training and validation sets
    - Create a new model
    - Train on training data
    - Predict on validation data
    - Print classification report and confusion matrix [9] [10] [12]
- At the end, print averaged performance metrics across all folds

average_reports(reports)
- Average all numeric values from classification reports across folds [6]
- Return a single averaged classification report

main()
- Load malicious URL dataset
- Parse each URL into its components using ‘split_url’
- Prepare input features and labels with ‘prep_inputs’ [5]
- Train and evaluate the model with ‘train_and_test’

4 Results

To test the model, 5-fold cross-validation was performed with each iteration training the model over 10 epochs. The results of each fold were recorded and the averages for all of the metrics were calculated. Overall, the CNN was able to achieve an accuracy of 97.8%. Looking at

the confusion matrix below (Fig. 1) it shows that the most difficult URLs to predict were phishing URLs, which the model incorrectly classified as benign.

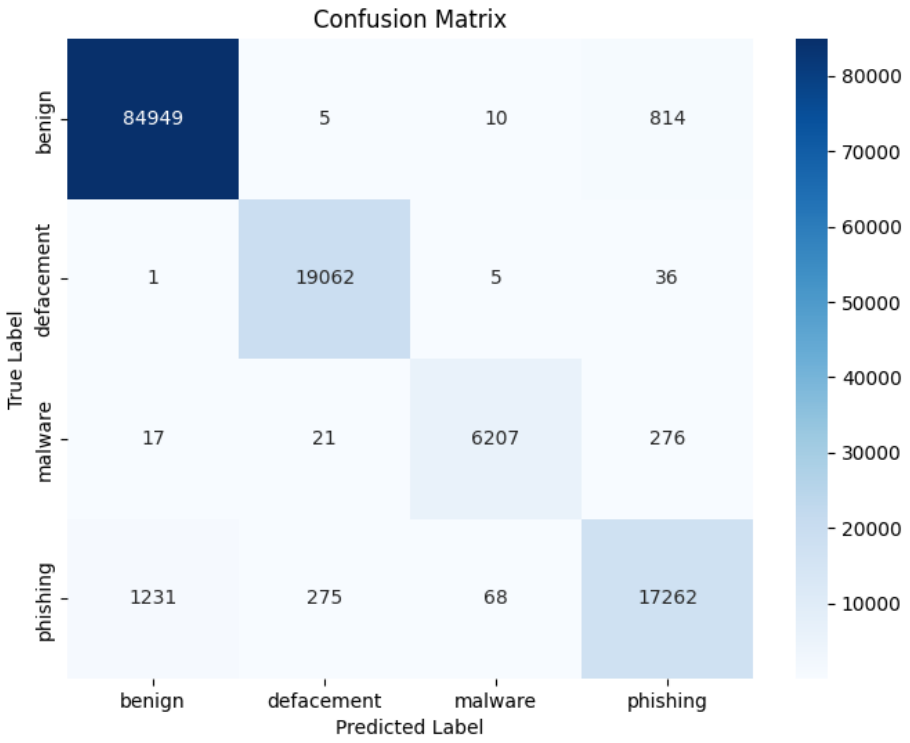

Fig. 1

Class	Precision	Recall	F1-Score	Support
Benign	0.982	0.992	0.987	85,620.6
Defacement	0.988	0.992	0.990	19,291.4
Malware	0.983	0.951	0.967	6,504.0
Phishing	0.943	0.907	0.925	18,822.2
Accuracy			0.978	130,238.2
Macro Avg	0.974	0.961	0.967	130,238.2
Weighted Avg	0.977	0.978	0.977	130,238.2

Table 1

Furthermore, taking a closer look at how the model performed, it can be seen from table 1 that the model performed very well when identifying the other types of URLs with defacement having the highest F1-score.

5 Discussion

The CNN created here achieves excellent performance across all URL classes, with an overall accuracy of 97.8%, macro average F1-score of 96.7%, and weighted average F1-score of

97.7%. Specifically, it demonstrates near-perfect precision and recall for benign and defacement URLs, and very strong results for malware and phishing URLs as well.

Looking back to the first paper (*"Classification of Malicious URLs Using Machine Learning"*), which found Random Forests to perform best with an F1-score around 92%, and Support Vector Machines achieving similar but slower results, the CNN outperforms in both accuracy and balance across classes. Moreover, while Random Forests struggled slightly more with phishing detection, the new CNN maintains a strong 92% F1-score for phishing URLs, indicating better generalization on the most challenging class.

Next, the second paper (*"Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions"*), highlighted traditional ML methods like Random Forests and SVMs reaching up to 95-99% accuracy. Additionally, the paper covered deep learning methods like CNNs and XGBoost which achieved up to 99.98% accuracy under ideal conditions. Comparing this to the CNN presented here, it can be seen that it matches or is only slightly behind previous methods. Despite scoring below the maximum of some other models, the CNN achieves high performance on a realistic dataset, indicating excellent real-world applicability without overfitting.

When compared to the third paper (*"Detection of Malicious URLs Using Machine Learning"*) which also explored classical and quantum machine learning methods, the CNN significantly outperforms the traditional models evaluated. The Decision Trees and SVMs achieved around 90–96% accuracy at best, while their quantum models (VQC) reached 97% only on a simplified, reduced dataset. The CNN achieves 97.8% accuracy on a full, large dataset, demonstrating an advantage in scalability and robustness compared to both classical and quantum approaches from that study.

Finally, regarding architecture, the CNN is more tailored to URL structure. It uses individual embeddings for subdomains, paths, queries, fragments, etc., separate convolutional layers for each component, global max pooling for dimensionality reduction, and dense layers with batch normalization and dropout for regularization and generalization. This makes it more specialized compared to other ML methods, explaining why the model performs well even across different attack types (defacement, phishing, malware).

Works Cited

[1] Abad S, Gholamy H, Aslani M. Classification of Malicious URLs Using Machine Learning. Sensors (Basel). 2023 Sep 8;23(18):7760. doi: 10.3390/s23187760. PMID: 37765815; PMCID: PMC10537824.

[2] M. Aljabri et al., "Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions," in IEEE Access, vol. 10, pp. 121395-121417, 2022, doi: 10.1109/ACCESS.2022.3222307.

[3] Reyes-Dorta, N., Caballero-Gil, P. & Rosa-Remedios, C. Detection of malicious URLs using machine learning. *Wireless Netw* 30, 7543–7560 (2024). https://doi.org/10.1007/s11276-024-03700-w

[4] Siddhartha, M. *Malicious URLs Dataset* (Version 1). Kaggle. https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset

[5] The pandas development team. (2020). *pandas-dev/pandas: Pandas* (Version 2.2.3) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3509134

[6] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link). (Version 1.26.4)

[7] Kurkowski, J. (2014). *tldextract: Accurately separate a URL's subdomain, domain, and public suffix.* [Python software]. https://github.com/john-kurkowski/tldextract (Version 5.1.3)

[8] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2016). *TensorFlow: Large-scale machine learning on heterogeneous systems*. arXiv preprint arXiv:1603.04467. https://arxiv.org/abs/1603.04467 (Version 2.19.0)

[9] Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830. (Version 1.6.1)

[10] J. D. Hunter, *Matplotlib: A 2D Graphics Environment* in *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, May-June 2007, doi: 10.1109/MCSE.2007.55. (Version 3.9.0)

[11] Python Software Foundation. (2023). *Python (Version 3.12)* [Computer software]. https://www.python.org

[12] Waskom, M. (2021). *Seaborn: Statistical data visualization*. Journal of Open Source Software, 6(60), 3021. https://doi.org/10.21105/joss.03021 (Version 0.13.2)