CS549 Machine Learning

Malicious URL Detection

Jianna Gapuz
Abigail Dupaya
James Duong
Bahnam Bahnam

In partial fulfillment for the requirements of the course CS549 Fall 2025
San Diego State University

Abstract

The current research concentrates on developing and comparing various approaches based on machine learning for efficient phishing URLs. The need for developing and testing multiple approaches made us rely on a combined dataset from the “Mendeley Phying URL Dataset”. A detailed preprocessing operation on the data was conducted with the objective of thoroughly training the models. It included balancing, eliminating inconsistencies, and determining vital features from a lexical as well as structural perspective, like the length of the URLs and special character utilization, and entropy. Four classification models were developed and compared among them: SVM, Adaptive Boosting, CNN, and RF. The efficiency of these models and approaches would be measured based on accuracy, precision, and F1-score, with a special focus on optimizing and enhancing the rate of recalling instances and lowering instances equivalent to false negatives. The efficacy of our research clearly establishes that ensemble and deep learning techniques, namely RF and CNN, perform significantly better compared to conventional methods pertaining to the detection of phishing threats.

Introduction

Background on Malicious URL Detection

The internet has become a vital part of modern society. Individuals, businesses, schools, government, healthcare, and critical services depend on the web to perform their daily operations. According to [1], this growing dependence has also increased the number of cyberattacks worldwide. One of the most common techniques used by cybercriminals is the creation of malicious URLs, which are web addresses designed to perform harmful actions when clicked by users. These actions can range from directing the user to a phishing website to downloading malware on their computer. According to the FBI’s 2023 Internet Crime Report, phishing and spoofing attacks resulted in almost $19 million in reported complaint losses, while malware accounted for over $1.2 million [2].

Typical approaches to detecting malicious URLs involve using blacklists and heuristic-based detection. Blacklisting works by comparing if a URL is in a list of known malicious URLs [3]. It is simple, but it cannot detect new or modified URLs that are not on the blacklist. Heuristic-based detection works by identifying if the URL exhibits patterns or signatures found in malicious URLs. Even though this approach can detect new unknown malicious URLs that blacklisting cannot, it often produces a high false positive rate [4].

URL Structure

Understanding the structure of a URL is crucial for correctly extracting lexical features that distinguish malicious URLs from legitimate ones. A Uniform Resource Locator (URL) is a string of text that refers to the unique address of a resource on the internet. Browsers use this string to figure out how and where to access the content. A URL is composed of the protocol, subdomain, domain name, top-level domain, port, path, query, parameters, and fragment [5].

The first part of the URL is the protocol, which tells how the browser communicates with the server. The most common protocols are HTTP (Hypertext Transfer Protocol) and HTTPS (Hypertext Transfer

Protocol Secure). HTTPS is the secure version of HTTP because it encrypts data during transmission between the browser and the server [6]. When present, the subdomain identifies a specific section within the main domain. Then it is followed by the domain name, which is the unique name of a website. The top-level domain indicates the type of content on the website [6]. For example, commercial websites use .com while educational institutions use .edu. The path specifies the exact location of the resource on the server [7]. If the URL is performing a query on the website, a question mark appears after the top-level domain. Out of all the parts of a URL, the protocol, domain name, path, and query are the ones used for malicious URL analysis, as they are the components modified by cybercriminals to deceive users into clicking a harmful link.

Feature Extraction Rationale

Feature extraction is the process of selecting relevant features from the raw data and converting them into numerical values that machine learning models can process. In malicious URL detection, the raw data is the URL, and the goal is to extract the meaningful features from this string and represent them numerically so that the machine learning models can learn from them. The features used to identify whether a URL is legitimate or malicious are length, special-character count, presence of specific keywords, and entropy, because cybercriminals usually manipulate these elements [8].

Overview of Dataset

The main dataset used in this paper is the Phishing URL Dataset from Mendeley Data [9]. There are two files in the dataset: Phishing URLs and URL dataset. The Phishing URLs file contains 54,807 phishing URLs, while the URL dataset file contains 104,438 phishing and 345,738 legitimate URLs. Both datasets have two columns, one for URL and one for the type. Since these two datasets have little overlap and have the same structure, they will be combined in order to increase the number of phishing samples and create a more balanced dataset.

Project Objectives

The objective of this project is to implement four machine learning models that accurately detect malicious URLs. The four models that were implemented are AdaBoost, Support Vector Machines, Convolutional Neural Networks (CNNs), and Random Forests. Data preprocessing and extraction of structural and lexical features was done to prepare the dataset for model training. Cross-validation was also done to determine the optimal hyperparameters for each model. After training, the models were evaluated and compared using performance metrics and comparative analysis.

Methodology

Data Preprocessing Steps

Preprocessing the datasets of URLs included a combination of typical preprocessing practices as well as annotating the data with information that may allude to a URL being malicious. For this project, our primary dataset was pulled from a Mendeley Dataset of Phishing URLs [9]. This data is consistent with its entries, for example, the inclusion of http/https schemes.

Balancing the Dataset

To ensure a balanced dataset, we will do a count of legitimate/malicious URLs in the dataset and check to see if they are balanced. A split of 50-50 of each label would be ideal, but the group settled on 70-30 to reflect the nature of the problem. In real life, malicious URLs are rarer in the dataset than benign ones.

Cleaning

1. Dropped exact duplicates in the dataset
2. Lowercased all letters (flagged URLs that contain uppercase letters)

Annotation

1. Used a parsing library (ie urllib.parse) to extract and append the following information:
    a. Scheme
    b. Subdomain
    c. Registrable domain
    d. Suffix
    e. Path
    f. Query
    g. Fragment
    h. Port
    i. Username
    j. Password
    k. Host
2. Added and flagged two new columns for http and https.
3. Count and append the following lengths:
    a. Total length
    b. Host length
    c. Path length
    d. Query length
    e. Fragment length
4. Counted and appended the counts the following characters:
    a. Dots
    b. Slashes
    c. Other special characters (ie -, _, %, @, ?, =, &)
    d. Digits
5. Calculated and appended the Shannon entropy of the URLs.
6. Flagged the following keywords:
    a. Login
    b. Verify
    c. Update
    d. Secure
    e. Account
    f. Bank
    g. Paypal
    h. Free
    i. Prize
    j. Gift
    k. Confirm
    l. Win
    m. Signin
    n. Support
7. Flagged link shorteners (ie bit.ly, t.co, tinyurl.com, goo.gl).
8. Flagged all domains that are purely IP numbers (ie http://101.10.1.101).

9. Dropped the fragment from the URLs

Model Selection and Justification

The four models that we selected to train are Support Vector Machine (SVM), Adaptive Boosting Classifier (AdaBoost), Convolutionary Neural Network (CNN), and Random Forests.

Support Vector Machine (SVM)

The SVM model provides a simple yet robust model for classification problems. It can employ strategies such as the kernel trick to add dimensionality to seemingly non-separable data. Since the given dataset is labeled, the SVM is a good choice for this problem. For this model, we employed Hinge loss and an L2 regularizer to minimize overfitting.

Adaptive Boosting Classifier (AdaBoost)

AdaBoost is an ensemble technique that combines weak learners to form a stronger classifier. It works by building the stumps sequentially, where each new stump will focus on correcting the misclassified samples of the previous stumps. Each new stump created has an estimator weight that measures how much say the stump will have in making the final classification.

In this implementation, a decision tree with only one node and two leaves, also known as a stump, was the weak learner. The model was chosen due to the characteristics of the features we extracted, its lightweight computational cost, its ability to achieve high accuracy, and it can improve recall for malicious URLs that are difficult to classify.

Since the project uses structural and lexical features, the dataset has several features that are non linearly separable. AdaBoost addresses this issue well because it naturally handles nonlinear data through the use of a stump as a base learner.

Convolutionary Neural Network (CNN)

The CNN model is lightweight and fast as it eliminates the need for manual feature extraction. It instead learns directly from the URL by applying 1D convolution and ReLU layers to capture local patterns. The process is then followed by global max pooling and a sigmoid output for classification. The model was trained using binary cross-entropy loss, and a L2 regularizer was used to reduce overfitting.

Random Forests

The implementation of the Random Forests was one of the supervised techniques conducted. This technique has been proven very effective as an ensemble learner. Its functionality involves the creation of numerous decision trees during the training phase. The output class is the mode of the classes obtained from each decision tree. This makes the technique very effective in the prediction of classes. This technique has been chosen since the total number of features we intend to extract appears to be quite high. Moreover, the technique provides the ability to assess the importance of the features. This could form the basis for the determination of the URL features that can predict the harmful nature.

To maintain alignment with the other models, a Random Forest classifier was developed using the group's standard set of features. This data contained structure-based features (for example, URL length, path length), lexicon-based features (for example, the number of special characters), and Shannon entropy. Following the group's approach, a 70/30 split was used for the training/testing data with stratified sampling, meaning that the number of samples per class was proportionally balanced between the training and testing sets. For Hyperparameter Tuning, we used GridSearchCV with 3-fold cross-validation. This helped us train the model on the highest possible recall, meaning that we get the lowest possible number of phishing URLs missed. The optimal hyperparameters that tuned the model were:

n_estimators: 100 (Amount of trees in the forest)

max_depth: None (To allow the trees to reach their fullest potential)

min_samples_split: 5 (preventing the model from splitting nodes with too few samples)

Training Process and Hyperparameter Tuning

Data Split

All the models used the same stratified train and test split with random state equal to 42 to ensure consistency and to provide a strong comparative analysis between all models. A stratified split was used because the dataset was imbalanced, consisting of 70% benign and 30% malicious URLs. Using stratification made sure that both the training and test sets preserved the same class distribution for benign and malicious URLs.

Hyperparameter Tuning

The models implemented different hyperparameter tuning techniques depending on the needs of the model. For the AdaBoost and Random Forest, both utilized GridSearchCV. This helped us train the model on the highest possible recall, meaning that we get the lowest possible number of phishing URLs missed. For the CNN, StratifiedKFold cross-validation was used to determine the optimal classification threshold. Because of the computational cost and complexity of the Quadratic Programming solver, we trained on a random subset of 5,000 samples using a linear kernel.

Performance Metrics

The statistical metrics we used to evaluate and compare our models were accuracy, precision, recall, and F1. While all of the point metrics are important, our primary focus was on optimizing recall. For identification, we used malicious URLs as the positive class, and benign (safe) URLs as the negative class. Here’s a breakdown of each metric, and what it means in the context of identifying malicious URLs effectively:

Accuracy measures how the proportion of URLs that the model correctly identifies is both malicious and safe. If accuracy is high, the model is making correct predictions. However, the amount of malicious URLs often outweigh the amount of benign URLs, making the metric misleading. A model could successfully label everything as safe and still have a high accuracy, while completely failing to detect actual threats.

$$Accuracy = \frac{TP + TN}{P + N} = \frac{TP + TN}{Total}$$

Precision measures how many of the URLs that were labeled malicious were actually malicious. Whenever a high precision model labels a URL as malicious, it's most likely correct, meaning it produces few false alarms, However, the metric doesn't tell you how many malicious URLs the model failed to detect, so it's possible to have high precision while still missing many real threats.

$$Precision = \frac{TP}{TP + FP}$$

Recall measures how many of the actual malicious URLs the model successfully identified. High recall means the model is capable of detecting most malicious URLs instead of letting them slip through. This makes the metric most suitable for this project, since missing even a single malicious URL can lead users to major risks.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{P}$$

F1 score is a metric that is a balance of both precision and recall. The metric tells us how well the model avoids false positives and how well it captures malicious URLs. This isn't the most ideal metric for the project since a good F1 score can still have poor precision/recall. Additionally, it's more important to successfully catch threats than maintaining a good balance between two metrics.

$$F1 = \frac{2 \times (Pr \times Re)}{Pr + Re} = \frac{2TP}{2TP + FP + FN}$$

Results

The table below demonstrates the precision, accuracy, recall, f1 score, and training time of the models.

Table 1. Performance Metrics for Malicious URL Detection models

Model	Accuracy	Precision	Recall	F1	Training Time (s)
SVM	0.9115	0.9907	0.7261	0.838	447.38
AdaBoost	0.9413	0.9191	0.8923	0.9055	26.59
CNN	0.9861	0.9998	0.9559	0.9774	233.57
Random Forest	0.9677	0.97	0.9307	0.948	64.81

The confusion matrix for each model was made to visualize how the predicted labels compared to the actual labels. Figure 1 to 4 shows the confusion matrix for SVM, AdaBoost, CNN, and Random Forest.

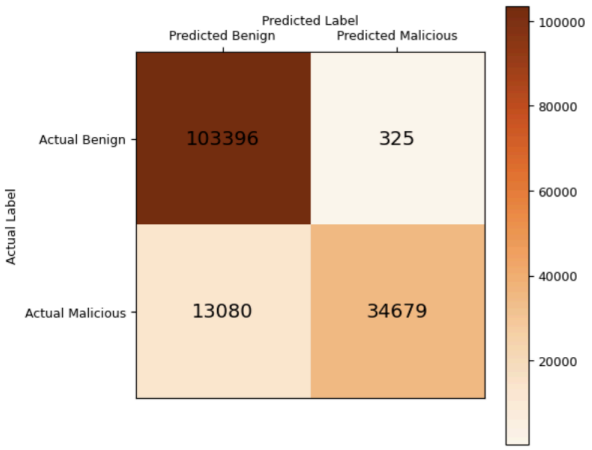

Figure 1. SVM Confusion Matrix

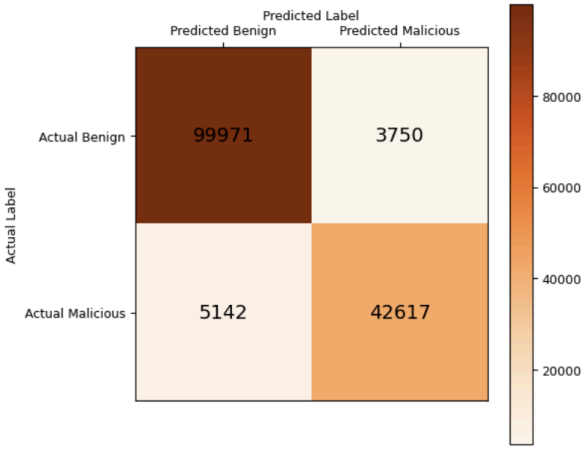

Figure 2. AdaBoost Confusion Matrix

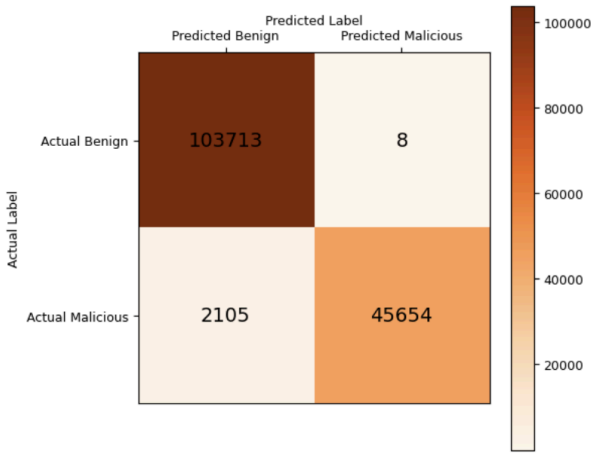

Figure 3. CNN Confusion Matrix

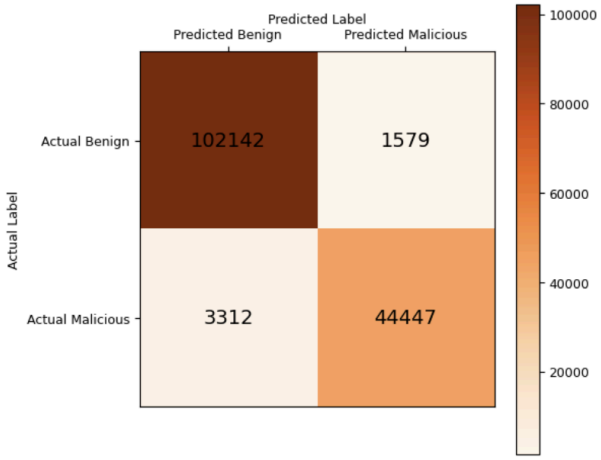

Figure 4. Random Forest Confusion Matrix

Discussion

SVM Discussion

The Support Vector Machine is a simple and robust solution to the URL classification problem due its simplicity to implement and ability to fine-tune features that best fit the model. The implementation of this model utilized the numeric features of the preprocessed dataset, using only 23 of the 35 features including http/https classification, total length, length of the host, path, query, and fragment portions, count of characters such as slashes, dots, digits, hyphens, underscored, percent symbols, at symbols, question marks, equal signs, ampersands and other special characters, keyword flagged, link shortener flagged, and IP host flagged.

Although the original plan was to split the training and testing data 70/30, the machine's RAM could only handle 5000 samples for the Q matrix. Despite this, the original plan to test on 30% of the dataset was

upheld. The memory limitation encountered in constructing the Q matrix demonstrates a constraint of classic SVMs on large datasets. QP-solved SVMs scale poorly with data size, which reinforces why linear or stochastic-gradient SVMs are preferred in industry for larger datasets.

An interesting finding was that the original plan of implementing Hinge Loss and an L2 regularizer resulted in a training time of roughly 1930.2119 seconds (32.17 minutes). The metrics of this model followed as:

Model	Precision	Accuracy	Recall	F1	Train Time
SVM with Hinge Loss and L2 Regularization	0.9184	0.9189	0.9189	0.9180	1930.2119 s

However, when Hinge Loss and L2 Regularization are removed, training time is decreased roughly by 76.8% only taking 447.384 seconds (7.46 minutes) to train. The evaluation metrics for this version of the model are as follows:

Model	Precision	Accuracy	Recall	F1	Train Time
SVM	0.9907	0.9115	0.7261	0.8380	447.384 s

The significant reduction in runtime can be attributed to the removal of the Hinge loss and L2 regularizer since both of these increase computational complexity of the solution. Without these components, the optimization relies on a simpler objective that allows the solver to converge more quickly. This change in runtime highlights the trade-off between computational efficiency and risk of overfitting. The reduction in recall and F1 scores suggest decreased robustness, specifically in detecting positive samples. A drop of 0.7% in accuracy, 19.28% in recall and 8.00% in the F1 scores for a ~76% improvement in training time may be a valid trade-off depending on the application. A lower recall means a higher rate of false negatives, where harmful URLs are incorrectly classified as safe, posing a real risk to users and systems. Therefore although the 76% drop in training time is significant, substantial drop in recall suggests that removing regularization may not be an acceptable trade-off in this specific scenario.

Adaptive Boosting Discussion

Adaptive Boosting is an ensemble method that combines multiple weak learners to form a strong classifier. It works well in situations where the positive class is rare, which aligns well with malicious URL detection, since malicious URLs occur less frequently than benign ones. This model demonstrated a strong balance between training time and performance.

In this implementation of AdaBoost, the base learners are made of decision tree stumps. Initially, an AdaBoost Classifier was trained using the default scikit-learn hyperparameters with 50 weak learners [10]. This baseline model trained very fast, taking approximately 6.67 seconds, and achieved a training accuracy of 92.04% and testing accuracy of 92.00%. The model also achieved a precision of 99%, but the recall was only 75%, which suggests even though the malicious predictions were usually correct, many malicious URLs were still classified as benign. This suggests that the baseline model was conservative in

labeling URLs as malicious. This is due to the imbalance of the dataset, the model focused on correctly identifying all the labels rather than focusing on identifying malicious URL samples.

Because the primary objective of this project is to detect malicious URL, improving the recall was the key priority. To address this limitation, hyperpameter tuning was done using GridSearchCV to improve the model's capability in identifying malicious URLs while maintaining reasonable precision.

The GridSearchCV explored different values for n_estimators, learning_rate, and estimator criterion. The best hyperparameters that was found was using gini as an estimator criterion, having a learning rate of 1, and 200 weak learners, which was used to build the final model. Although the training time increased compared to the baseline model, the additional cost was small and justified by the improvement of the performance. The decision threshold was also set to 0.48 instead of the standard 0.50 to prioritize recall, allow the model to classify borderline cases as malicious and reduce the risk of undetected malicious URLs. While this change led to a decrease in precision, it significantly improved recall which makes the model more effective for malicious URL detection.

After applying hyperparameter tuning and adjusting the decision threshold, AdaBoost achieved an accuracy of 94.13%, a precision of 91.91%, recall of 89.23%, and an F1 score of 90.55%. Out of all the 151,480 samples, the model correctly identified 42,617 malicious URLs as malicious, while 5,142 malicious URLs were misclassified as benign. This shows that the model can still be improved in terms of recall.

Model	Precision	Accuracy	Recall	F1	Train Time
AdaBoost	0.9191	0.9413	0.8923	0.9055	26.59 s

AdaBoost was easy to train and the only challenge faced during the process was the data imbalance. Since the model uses decision tree stumps as base learners, the model favored the majority class which was benign. It resulted in high performance when predicting benign but weaker on malicious URLs. To resolve this problem hyperparameter tuning was done and the decision threshold was lowered, so that the model can prioritize recall and reduce the number of missed malicious URLs.

CNN Discussion

The CNN used in this project is a 1D convolutional model designed for character-level URL data. A 1D CNN was appropriate for this task because it can automatically detect short, meaningful patterns in the character sequence that often indicate malicious behavior. To ensure all URLs could be processed uniformly, each URL was encoded to a fixed length with padding if necessary. Using a kernel size of 5, the convolution layers slide across the URL and capture local 5-character patterns, while pooling layers reduces dimensionality and retain the only important features. These features are then passed into dense layers, which combine the learned representations and generate a final classification output.

To improve the model, stratified k-fold cross-validation was used. The model was trained multiple times on different splits of data, while keeping the class balance the same. After each fold, a separate function was called to find the best prediction threshold to maintain a strong recall. The final threshold used was the average of the thresholds amongst all the folds.

After tuning, training, and evaluation, the model produced an accuracy score of 98.61%, a recall score of 95.59%, a precision score of 99.98%, and a F1 score of 97.74%. Out of the 151,480 samples, the model produced 103,713 true negatives, 8 false positives, 2,105 false negatives, and 45,654 true positives. Due to threshold tuning the model was able to exceed the targeted recall threshold of 90%, and recall was targeted due to its importance in security-driven classification tasks.

Model	Precision	Accuracy	Recall	F1	Train Time
CNN	0.9998	0.9861	0.9559	0.9774	233.5655 s

Several challenges were encountered throughout the implementation and tuning of the CNN model. Although the training time of the CNN was only 233 seconds, it required 599 seconds for hyperparameter tuning. 3 splits and 3 epochs were used during the tuning stage, primarily because the tuning process became computationally expensive. The amount of time it took in total slowed down the overall experimentation cycle, limiting how quickly adjustments could be tested and evaluated.

Another challenge was the complexity of preprocessing the data, because the only features CNN used were the URL type and raw URL string itself. Before processing the data, it needed to be encoded by converting every unique character into a corresponding numeric index. This required building a comprehensive character/index dictionary and ensuring that all URLs were padded or truncated to a uniform length. This was only challenging because I initially didn't realize encoding was necessary for a CNN to interpret the data effectively.

Random Forest Discussion

The Random Forest model showed high efficacy in identifying Legitimate vs. Phishing URLs, with an overall accuracy of 96.77%. An interesting aspect in our analysis was the Recall of the model for the phishing class to be around 93.07%. In cybersecurity, recall is more important compared to precision because the cost of a missed malicious URL (False Negative) is way costlier compared to a safe one being flagged off. The model is able to identify 44,447 phishing, while it missed 3,312 as False Negatives.

The Precision of 97.00% is very high, reflecting the low count of False Positives at only 1,579. This means that the model is not over-aggressively flagging benign sites-a very important feature for retaining user trust in any real-world deployment. It was found that allowing trees to grow fully, with a moderate number of estimators, gave the best trade-off between bias and variance for this dataset, capturing the complicated, non-linear interactions in URL structures without overfitting.

Model	Precision	Accuracy	Recall	F1	Train Time
Random Forests	0.97	0.96	0.93	0.948	64.81 s

The important features were: is_https/is_http (0.46 total): That is, the presence or absence of secure protocols was the strongest predictor.

entropy_url (0.093) - Randomness of the characters within the URL that strongly indicated phishing pages.

count_dots(0.092) - Count too many dots. This is typically used in subdomain obfuscation. This was one of the indicators.

Cost of Computation of the Ensemble Methods: Another important challenge was computation cost for hyperparameter tuning. Unlike simpler models, training hundreds of decision trees was computationally expensive in the GridSearchCV process, taking ~528 seconds to converge. This can be understood as a trade-off between the high accuracy of Random Forests and the time efficiency of training compared to lighter models.

Sensitivity to Feature Engineering: Initial iterations of the model based on raw URL data produced significantly worse recall, around 60%, as was observed in early testing. This model had difficulty generalizing from only simple lexical features. This was resolved by using the standardized dataset from the group, processed_urls.csv, which contained high-level engineered features such as is_https, entropy calculations, and counts of specific characters. This underlined the fact that, in the case of Random Forests, the quality of feature extraction is equally as important as the algorithm itself.

Comparative Analysis

The four models showed strong performance in predicting malicious URLs, but they differed in their ability to balance recall, precision, and training time. The models focused on increasing recall since misclassifying malicious URLs as benign is more harmful than predicting a benign URL as malicious. For this reason, the models were prioritized to be more aggressive rather than conservative. The CNN and Random Forest achieved the highest recall values, with 95.59% and 93.07%, respectively. This makes these models more suitable in cases where security is of utmost importance, as they generate fewer false negatives. The fastest model was AdaBoost, which only took 26.59 seconds to train, followed by Random Forest, which took 64.81 seconds. Although AdaBoost trained the fastest, its recall was 6.36% lower than the CNN, which achieved the highest recall. On the other hand, while the CNN achieved the best recall, it had a longer training time and was 206.98 seconds slower than AdaBoost. In terms of the trade-off between training time and recall, the Random Forest provided the best balance, as it trained relatively quickly, only taking 64.81 seconds, while still achieving a high recall. Of the two ensemble methods done in this project, the bagging method employed by Random Forest performed better because training trees independently reduced variance and made the model more robust to noise and class imbalance compared to AdaBoost's sequential boosting approach. In terms of precision, the SVM performed the best with a precision of 99.07% but it suffered from the lowest recall at 72.61% and took the longest to train, which is 447.384 seconds or approximately 7.46 minutes.

Conclusion

List of Deliverables

The project proved the successful applicability of machine learning algorithms to detect malicious URLs without examining their contents. We were able to build four models using feature extraction obtained from raw URLs without manually examining them: a SVM model, an AdaBoost model, CNN, and a Random Forest model, and tested them on accuracy for classification as a legitimate and phishing link. Results show that Convolutional Neural Network and Random Forest provide better accuracy, with more than 93% recall value. It confirms our hypothesis that highly nonlinear and complex attributes associated

with raw URLs, taking into consideration high entropy and specific numbers of special characters, are good indicators for identifying malicious URLs. The Random Forest model worked as a highly efficient technique with accuracy as 96.77%, with low value and well-interpret-able. It identified attributes relating to security and randomness associated with the phishing link and structure.

Overview of Issues Faced

During its development, several key technical challenges were addressed. First, it became evident that there were some limitations with regards to scalability within the SVM implementation. Although there were limitations on memory capacity, they greatly limited the size of the training data. Secondly, there were some problems with regards to class imbalance within the AdaBoost model, which eventually had to undertake extensive thresholding. Thirdly, character-level encoding and preprocessing within the deep learning approach created some dilemmas for the CNN, while feature engineering made it expensive and highly sensitive within the Random Forest.

Future Work Suggestions

Some possible paths for future work on this project would be incorporating functionality based on content, maybe looking at HTML tags or leveraging knowledge of JavaScript, as might assist with the detection of “cloaked” attacks with a benign-looking string contained within the url. It would be necessary for these models and others like them to be tested with adversarial samples, or urls that have intentionally been modified as a method for thwarting machine learning-based classification.

Individual Contributions

Task division

The team decided to divide the tasks based on the weight and the time required to complete them. For each task assigned to the member, they are responsible for implementing the code and writing the documentation for the final report.

Final Report Individual Contributions

Jia Gapuz

- Cleaned, extracted, annotated, and encoded the datasets used in training.
- Trained, optimized, and discussed the results of SVM.
- Create a GitHub repository to compile all the project code in one location.
- Create the README file, which contains the instructions to run the code and details on the environment in which the models are running.

Abigail Dupaya

- Trained, optimized, and discussed the results of AdaBoost.

- Created the table and confusion matrix visualization showing the performance of all the models and provided the comparative analysis of all the models performances.
- Wrote and updated the introduction, which included the problem statement, background information, overview of the dataset, and project objectives.

James Duong

- Responsible for coding and documenting the evaluation metrics used in the project.
- Train, optimized, and discussed the results of the CNN model
- Identified and justified the evaluation metrics used to assess the performance of the model.

Bahnam Bahnam

- Wrote and updated the abstract, which contains a concise summary of the project objectives, models developed, results of the models developed, and the conclusion.
- Trained, optimized, and discussed the results of Random Forest.
- Discuss the improvements that can be made for future work.

References

[1] E. Lee, “More dependence on internet leads to more cyberattacks worldwide,” *Voice of America*, Aug. 26, 2017. [Online]. Available: https://www.voanews.com/a/dependence-on-internet-leads-to-more-cyberattacks/4001728.html

[2] Federal Bureau of Investigation, “2023 Internet Crime Report.” [Online]. Available: https://www.ic3.gov/Media/PDF/AnnualReport/2023_IC3Report.pdf

[3] “View of An Identification and Analysis of Harmful URLs through the Application of Machine Learning Techniques.” https://ijisae.org/index.php/IJISAE/article/view/4905/3595

[4] D. Bhadani, “Heuristic-Based Phishing Site Detection,” MS Thesis, California State University Northridge, 2023. [Online]. Available: https://scholarworks.calstate.edu/downloads/v692td67g

[5] J. Pyram, “9 Parts of a URL You Should Know,” *Medium*, Oct. 09, 2020. https://medium.com/@joseph.pyram/9-parts-of-a-url-that-you-should-know-89fea8e11713

[6] Cloudflare, Inc., “What is HTTPS?,” *Cloudflare*. https://www.cloudflare.com/learning/ssl/what-is-https/#:~:text=Hypertext%20transfer%20protocol%20secure%20(HTTPS,web%20browser%20and%20a%20website. (accessed Oct. 28, 2025).

[7] Zvelo, “The anatomy of a full path URL: hostname, protocol, path, and more…,” *Zvelo*, Oct. 28, 2025. https://zvelo.com/anatomy-of-full-path-url-hostname-protocol-path-more/

[8] M. Mamun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, and A. Ghorbani, “Detecting Malicious URLs Using Lexical Analysis,” in *Lecture Notes in Computer Science*, vol. 9955, Springer International Publishing, pp. 467–482. doi: 10.1007/978-3-319-46298-1_30.

[9] J. K. S. Kaitholikkal and A. B, “Phishing URL dataset,” *Mendeley Data*, Apr. 2024, doi: 10.17632/vfszbj9b36.1.

[10] Pedregosa *et al.*, “SciKit-Learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Nov. 2011.