Project Definition

Project Overview

Phishing is a widespread cyber threat, with attackers often creating deceptive websites that closely resemble legitimate ones. These fraudulent sites trick users into sharing sensitive information such as login credentials or payment details. According to the Anti-Phishing Working Group (APWG), phishing attacks surged to an all-time high in 2023, affecting individuals and organisations across all sectors [1].

Traditional rule-based systems to detect phishing are often too rigid and can be easily circumvented by attackers who craft URLs that closely mimic trusted domains. Machine learning offers a more dynamic approach, as it can learn patterns in URLs and identify those that are likely to be phishing attempts. Research by Sahoo et al. (2017) [2] highlights how machine learning models trained on lexical and structural URL features can generalise well to unseen phishing attacks.

The aim of this project is to build a machine learning model that can classify URLs as either phishing or legitimate. This model will be deployed as a real-time prediction service using AWS. The dataset for this project comes from the UCI Machine Learning Repository - PhiUSIIL Phishing URL dataset[3], which contains 235,795 labelled URLs and a range of features derived from them. The label is binary, 0 for phishing or 1 for a legitmate URL.

To ensure that my model is consistent and reliable during real-time inference, I have written custom code to engineer these features myself. While I matched the original feature names, I created my own logic to compute them because the exact feature calculation methods used in the dataset were not available. This guarantees that the features can be computed accurately for any new URL during inference.

Problem Statement

The challenge is to develop a binary classification model that can determine whether a URL is phishing or legitimate. The model must be fast and accurate so that it can be deployed as an endpoint that users or applications can query in real-time. The solution must accept a URL as input and return a clear and confident prediction of phishing or legitimate.

To achieve this, I will use AWS services to train and deploy the model, ensuring that the entire process from URL submission to prediction is streamlined. There should be no need for manual feature extraction or preprocessing by the user, making the solution practical and easy to integrate into existing systems. Because phishing has serious security and financial implications, the model must achieve high precision to avoid false alarms and high recall to ensure that real threats are not missed.

Metrics

To evaluate how well the model performs, I will track the following metrics:

- Accuracy: The overall proportion of correctly predicted URLs.
- Precision: The fraction of predicted phishing URLs that are truly phishing. This is crucial to avoid unnecessary blocking of legitimate sites.
- Recall: The fraction of actual phishing URLs that were correctly detected, ensuring that as many threats as possible are identified.
- F1-score: A balanced metric that combines precision and recall, useful when both types of errors are important.
- Log-loss: A measure of how confident and accurate the model's probability predictions are. Lower log-loss indicates better performance.
- Confusion Matrix: A breakdown of true positives, false positives, true negatives and false negatives to give a clear picture of how the model performs in different scenarios.

These metrics will ensure that the model is not just accurate but also balanced in catching phishing attempts while minimising false alarms.

Analysis

Data Exploration

As previously discussed, the dataset comprises numerical features extracted from the URLs themselves. The original dataset did not disclose its feature engineering methodology, so I implemented my own logic to calculate the same features. This ensures that I can consistently and accurately generate feature values for any new, unseen URLs during real-time inference.

The full set of engineered features includes:

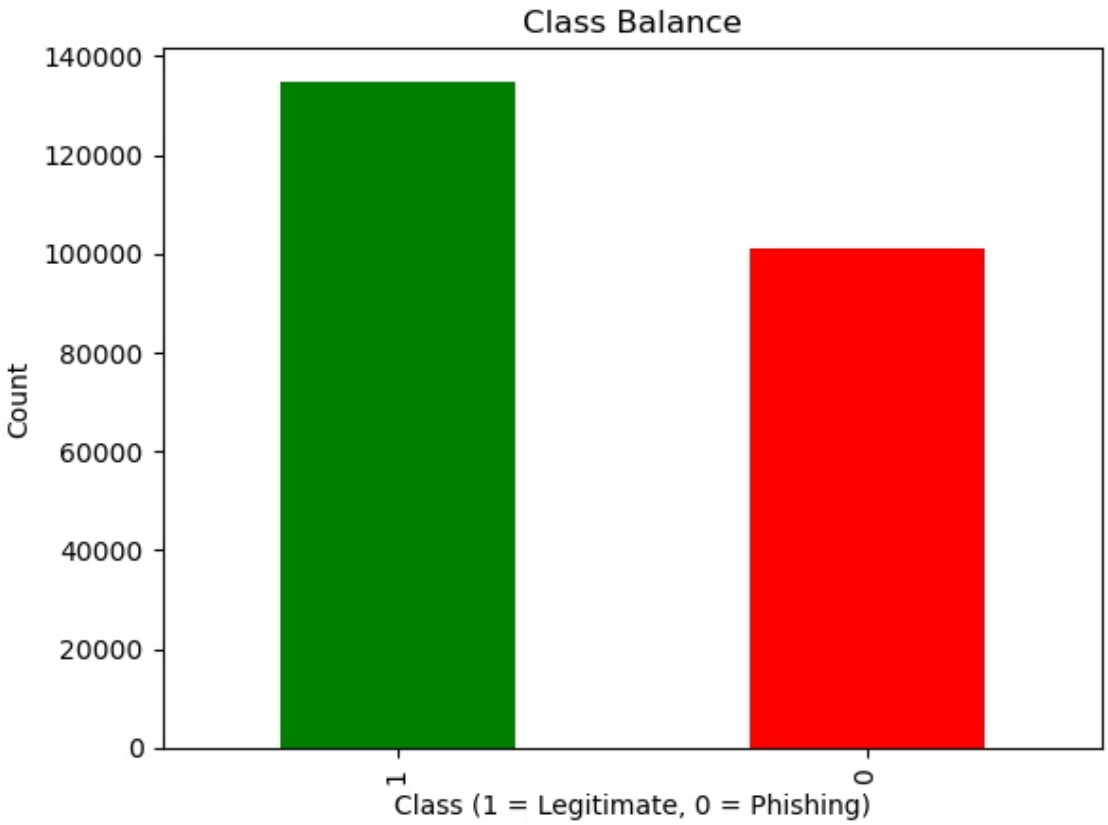
- URL (the full URL itself)
- URLLength
- DomainLength
- IsDomainIP
- NoOfSubDomain
- NoOfLettersInURL
- LetterRatioInURL
- NoOfDegitsInURL
- DegitRatioInURL
- NoOfEqualsInURL
- NoOfQMarkInURL
- NoOfAmpersandInURL
- NoOfOtherSpecialCharsInURL
- SpacialCharRatioInURL
- IsHTTPS
- CharContinuationRate

Exploratory Visualisation

A key observation during exploration was the distribution of the target variable:

- Legitimate (1): 57.19%
- Phishing (0): 42.81%

This target balance is above the 40% threshold suggested by Google's ML Crash Course on imbalanced datasets, so no resampling or rebalancing techniques were required [4]. This can also be seen in the plot below



Further visualisations were generated to understand feature distributions and correlations. These will be discussed in Methodology section in the report as they are heavily tied with that section.

These visual insights were used to guide feature selection and to identify any potential collinearity issues within the feature set.

Algorithms and Techniques

For this binary classification problem, I chose to use XGBoost as the primary model. XGBoost was selected for its ability to handle tabular, numerical data and for its robustness to collinearity and potential noise in feature sets. It also offers:

- Flexibility in regularisation to avoid overfitting
- Scalability in large datasets and parallelism for efficient training
- Excellent compatibility with AWS SageMaker for seamless deployment

Benchmark

To establish a baseline, I trained a simple logistic regression model using only the URLLength

feature. Since phishing URLs are often longer than legitimate URLs, this provided a natural and interpretable starting point for evaluating performance.

The logistic regression benchmark achieved the following performance:

- Accuracy: 0.7425
- Precision: 0.7201
- Recall: 0.8994
- F1 Score: 0.7998

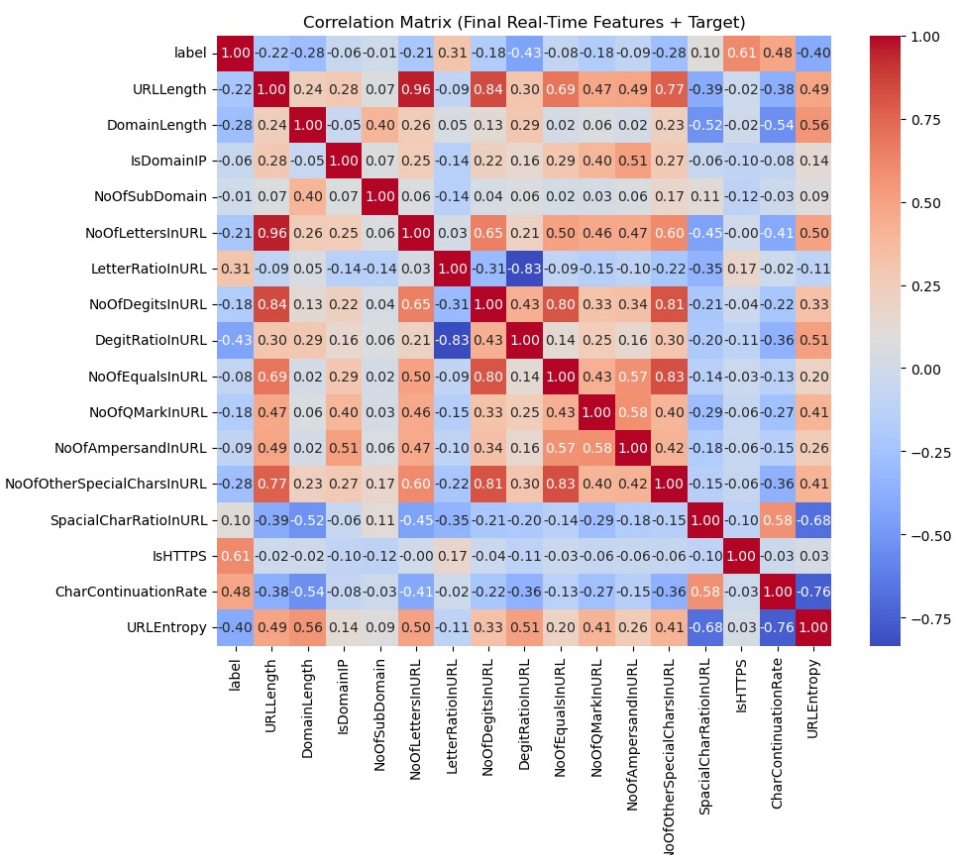
This provided a solid benchmark to improve upon with the more advanced XGBoost model trained on the complete feature set.

Methodology

Data Preprocessing

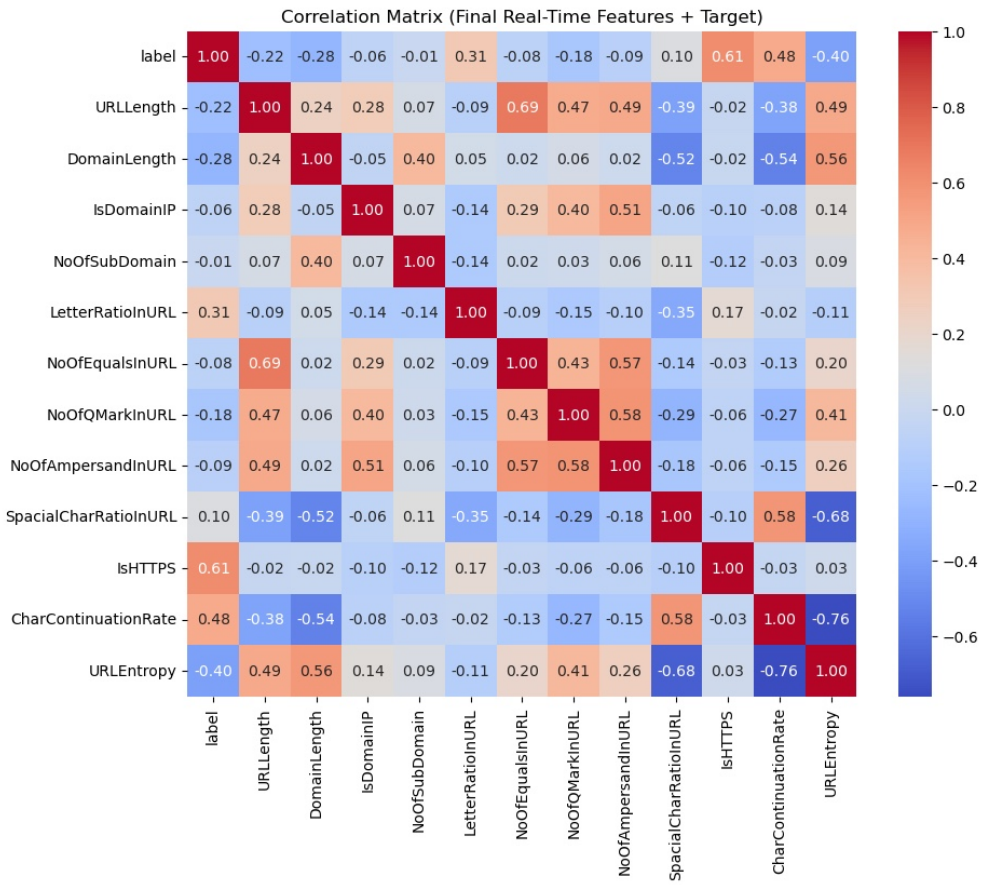
For this project, I developed a custom feature engineering pipeline that generates numerical features directly from the URL string. This was crucial for two reasons: it ensured that we could consistently compute these features during inference on new, unseen URLs, and it removed any dependencies on dataset-specific logic that was not publicly available.

The custom feature engineering logic includes metrics such as URL length, domain length, the ratio of letters, entropy (as a measure of randomness in the URL), and the number of subdomains, among others. The inclusion of entropy was based on the intuition that phishing URLs often have more randomness to evade detection.



A correlation analysis was performed to identify multicollinearity and redundancy between features. This analysis revealed strong correlations between several features, including NoOfLettersInURL, NoOfDegitsInURL, NoOfOtherSpecialCharsInURL, and DegitRatioInURL. To address this, I decided to drop these features from the dataset. This reduced redundancy and simplified the feature space, making the final model more efficient for real-time inference. This is shown above, as the features with correlations above 0.8.

I then verified the correlations again to ensure that the remaining features did not have significant multicollinearity. The correlation matrix after this is shown below.



After finalising the feature set, I split the cleaned dataset into training, validation, and test sets using a 70:15:15 split. This approach ensures that the model can monitor itself during training while leaving a hold-out set for unbiased final evaluation.

Implementation

For model implementation, I used Amazon SageMaker's built-in XGBoost algorithm. The XGBoost container was selected because it is well-suited to binary classification tasks and is optimised for tabular data with numerical features. It also scales well and has built-in support for distributed training.

The hyperparameters I selected were:

- max_depth: 5

- eta: 0.2
- gamma: 4
- min_child_weight: 6
- subsample: 0.7
- objective: binary:logistic
- num_round: 300

Although I set up multi-instance training with fully replicated data, this was mainly for demonstration purposes to illustrate future scalability, as it was not strictly necessary for this dataset's size.

Complications

During the migration to the latest XGBoost container version (1.7-1), I encountered errors regarding the label column's location. After investigating using CloudWatch logs and consulting the SageMaker documentation, I discovered that this version of the container requires no header in the dataset and expects the label to be the first column, not the last. Once I resolved this, the training proceeded smoothly.

Model Performance

Using the test dataset, the model achieved the following performance:

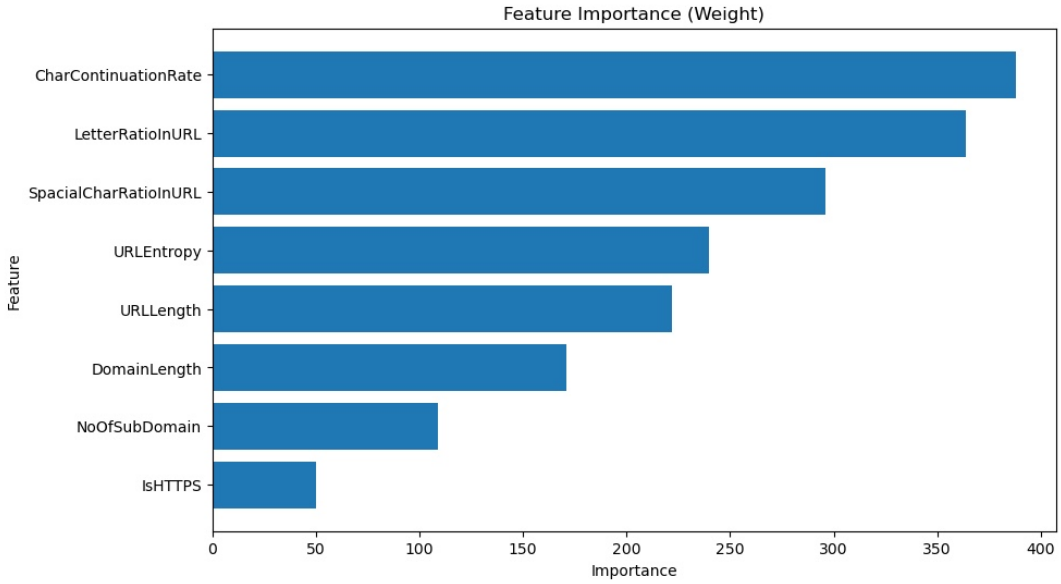
- **Accuracy: 0.9953**
- Confusion Matrix:

  [[15008 134]
  [32 20195]]
- Classification Report:

	precision	***recall***	***f1-score***	***support***
0	***1.00***	***0.99***	***0.99***	***15142***
1	***0.99***	***1.00***	***1.00***	***20227***

Refinement

To further refine the model and reduce the complexity of the feature set, I analysed the feature importance scores from the trained model using the built in xgboost feature importance tool. This analysis revealed that features **IsDomainIP, NoOfEqualsInURL, NoOfQMarkInURL, and URLEntropy,** contributed little to predictive performance as shown by the below feature importance plot.



Dropping these features streamlined the model for faster inference and reduced the amount of feature engineering required at runtime.

I retrained the model with the reduced feature set and achieved similar high performance:

- **Accuracy: 0.9954**
- Confusion Matrix:

  [[15012 130]
  [33 20195]]
- Classification Report:

	precision	***recall***	***f1-score***	***support***
0	***1.00***	***0.99***	***0.99***	***15142***
1	***0.99***	***1.00***	***1.00***	***20228***

This confirmed that the model maintained high performance with fewer features.

Deployment

To enable real-time predictions, I deployed the final model as a SageMaker endpoint. To ensure that URLs could be passed in their raw form without any prior preprocessing, I implemented a Lambda function to compute the necessary features dynamically. This Lambda function extracts the required features and calls the endpoint, returning the prediction in an easily consumable format.

I also ensured that the Lambda function had the appropriate permissions to invoke the SageMaker endpoint, completing the end-to-end deployment pipeline.

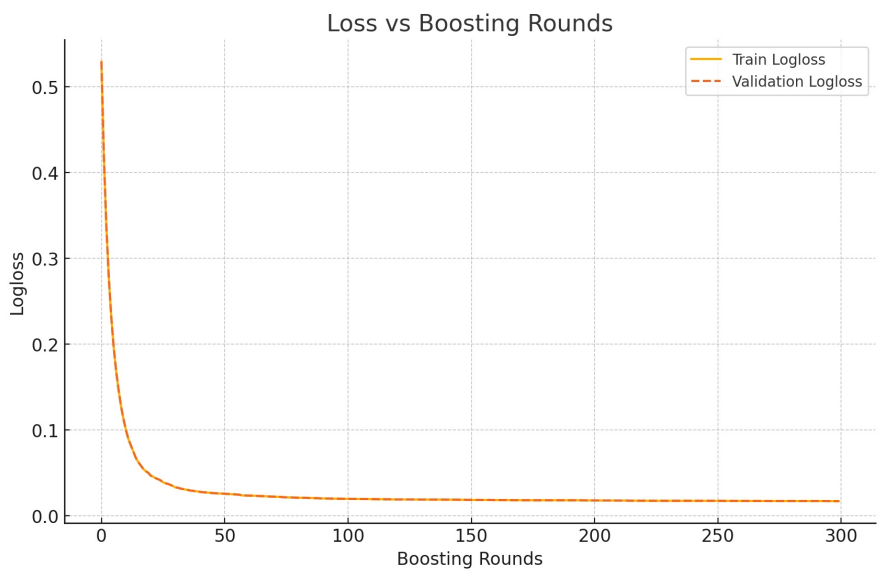
RESULTS

Model Evaluation and Validation

The final model was an XGBoost classifier, trained using the following hyperparameters:

- max_depth: 5
- eta: 0.2
- gamma: 4
- min_child_weight: 6
- subsample: 0.7
- objective: binary:logistic
- num_round: 300

These hyperparameters were selected as they represent a reasonable starting point for binary classification tasks. The max_depth of 5 ensures that trees are not too complex, which helps reduce the risk of overfitting. The eta learning rate of 0.2 balances convergence speed and stability. gamma and min_child_weight encourage more conservative splits, reducing the risk of overly-specific decision boundaries. The subsample parameter controls the fraction of observations used to build each tree, improving generalisation. The objective is set to binary:logistic as the problem is binary classification, while num_round ensures the model trains long enough to fully converge, in this case 300 rounds as can be seen by the below plot of the loss vs boosting rounds for the XGBoost models.



To accurately assess the robustness of the model, a hold-out test set comprising 15% of the dataset was created at the outset. This data was not involved in any stage of training or hyperparameter tuning. By evaluating the final model on this test set, we obtained an unbiased assessment of its performance and ability to generalise to unseen data. This is considered good practice to minimise bias and ensure model reliability.

Final Results:

Model Variant: Logistic Regression (benchmark)
Accuracy: 0.7425
Final Validation Log-Loss: N/A
Notes: Simple baseline, only URLLength

Model Variant: XGBoost (300 rounds, full features)
Accuracy: 0.9953
Final Validation Log-Loss: 0.01907
Notes: Full convergence, full feature set

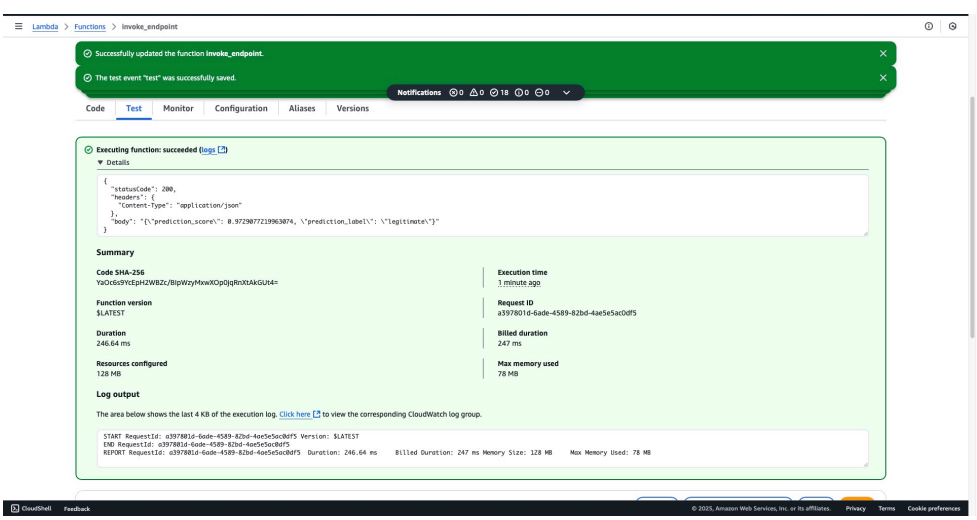
Model Variant: XGBoost (300 rounds, reduced features)
Accuracy: 0.9953
Final Validation Log-Loss: 0.01876
Notes: Final model, reduced feature set

It is evident from the table above that the final XGBoost model significantly outperforms the benchmark logistic regression model, achieving an accuracy of approximately 0.9953 compared to 0.7425 for the baseline. This demonstrates a substantial improvement in performance.

[[15012 130]
[33 20195]]

This matrix illustrates how few false positives (legitimate URLs incorrectly flagged as phishing) and false negatives (phishing URLs incorrectly classified as legitimate) remain. Such a low rate of misclassification is particularly important in phishing detection, where false negatives can lead to security breaches and false positives can degrade user experience.

A key requirement for this project was achieving low latency. This was successfully met: the deployed endpoint, hosted on an ml.m5.large instance and invoked via AWS Lambda, achieved an average inference latency of just 246 ms as shown by below screen shot.



Justification and Significance

The final results, with high accuracy, precision, and recall, coupled with minimal false positives and false negatives, validate that the model meets the original project goal of reliably detecting phishing URLs. The significant performance improvement over the baseline logistic regression model—particularly in terms of F1-score and overall accuracy—demonstrates that the chosen XGBoost model is not only theoretically sound but also practically viable.
Given just a URL string as input, the model can determine in real time whether it is a phishing attempt or a legitimate site, making it an effective tool for organisations or users seeking to reduce their exposure to phishing threats.

References

References: [1] APWG. (2023). Phishing Activity Trends Report. https://apwg.org
[2] Sahoo, D., Liu, C., & Hoi, S. C. (2017). Malicious URL Detection using Machine Learning: A Survey. ACM Computing Surveys, 50(3), 1–33.
[3] Dua, D. & Graff, C. (2019). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. Retrieved from https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset
[4] Google's ML Crash Course on Imbalanced Datasets. https://developers.google.com/machine-learning/crash-course/overfitting/imbalanced-datasets..