# Optimizing URL Phishing Detection: A Manifold Learning Approach with an Efficient Neural Network Focused on Reducing Computational Cost

Vijay Venkatesh M
*Department of Computer Science*
*University of Madras*
Chennai, India
vijayvenkatesh2212@gmail.com

***Abstract*—The growing number of phishing attacks is one of the top concerns of cybersecurity researchers. Cryptographic methods are not reliable in stopping phishing attacks because they manipulate the users into thinking it is safe to access a web link. This results in compromise of sensitive information. Recent research has shown how machine learning methods are capable of detecting malicious websites. However, machine learning models can be computationally heavy to produce good accuracy. This paper focuses on reducing the computational cost while maintaining an effective prediction model. The experiment is conducted on a dataset of size (30, 10000), consisting of phishing and benign website links. The Isomap algorithm is implemented to reduce the input dimension while preserving correlation. Subsequently, a neural network is employed, utilizing simple activation and optimization functions to ensure efficient and rapid prediction. The model achieves an accuracy of 87% under 30 training epochs.**

***Keywords— Phishing, Manifold Learning, Neural Networks, Computational Cost, Optimization.***

## I. Introduction

The growth of the internet has been tremendous in recent times, making it the world's largest and fastest-growing dynamic network. With over 5 billion internet users each day, the internet has become ubiquitous. We utilize web services for various activities, such as knowledge sharing, social communication, and conducting financial transactions. Unfortunately, the internet is also a breeding ground for unwanted content, including phishing, drive-by downloads, host drive-by exploits, and spam [1].

Considering the high level of user activity on the internet, phishing stands out as one of the most common types of social engineering attacks targeting the vulnerabilities of system users [2]. It creates an illusion of integrity in the URLs and persuades the users to take actions that compromise their sensitive information, such as passwords, identity details, social security numbers, or any such valuable information [3]. According to IBM's 2022 Cost of Data Breach Report, phishing is the second-largest and most expensive cause of compromising information, resulting in an average cost of $4.91 million for responding organizations [4]. CISCO reports that, in 2021, the probabilistic percentage of at least one individual clicking a phishing link from an organization was about 86% [5].

In the second quarter (Q2) of 2022, APWG observed a total of 1,097,811 phishing attacks, making a new record and the worst quarter for phishing APWG has ever observed. The average requested amount for Business Email Compromise (BEC) attacks on remittances in Q2 2022 was $109,467, up from $91,436 in Q1 2022 [5]. The healthcare and transportation industries suffered an increase in ransomware attacks in Q2 2022. Mobile-based fraud has also risen, with smishing and vishing increasing in Q2 2022 [5].

Numerous approaches have been employed to filter out phishing websites. Such as server-side filters, authentication, protection, and user education [6]. While there are distinctive characteristics in these attacks, leading researchers to deploy machine learning techniques. In recent years, scientists have developed and applied various algorithms to predict URL phishing attacks, including computationally intensive models such as Deep Neural Networks, Ensemble Methods, and Natural Language Processing Transformers. This paper aims to explore a more computationally efficient approach for predicting phishing URLs. Given constraints such as limited data, the proposed method involves narrowing the feature space and constructing a simplified neural network model with fewer layers and nodes.

## II. Literature Review

There exists a substantial body of recent literature dedicated to phishing detection, with a focus on leveraging machine learning, deep learning, and natural language processing (NLP) techniques. techniques. Researchers have investigated various methods to enhance the effectiveness of phishing detection systems, recognizing the escalating sophistication of phishing attacks. Machine learning approaches have been extensively employed for URL-based phishing detection, with studies utilizing traditional classifiers like support vector machines (SVM), decision trees, and random forests.

Deep learning techniques, especially neural networks, have gained attention due to their inherent capability to automatically learn hierarchical representations from data. Notable research endeavors include the application of convolutional neural networks (CNNs) to effectively capture spatial and temporal patterns in URL and website content, thereby enhancing the discriminative power of phishing detection models [7]. Recent research has investigated the utilization of Natural Language Processing models to encode URLs, signifying a discernible trend towards leveraging advanced language models for improved phishing detection [8]. This approach aims to capture semantic information within URLs [9]. In the pursuit of comprehensiveness and clarity, researchers have consistently explored novel methodologies to address the evolving landscape of phishing attacks. Advanced techniques, such as convolutional neural networks and Natural Language Processing models, showcase promising capabilities in capturing intricate patterns and semantic nuances, reflecting the dynamism and sophistication of contemporary phishing threats.

TABLE I. SUMMARY OF RELATED WORK

| *Ref* | *Algorithm* | *Accuracy* |
|---|---|---|
| [9] | CatBoost | 96.9% |
| [9] | XGBoost | 92.1% |
| [9] | Light GBM | 95.2% |
| [10] | PUCNN | 95.7% |
| [7] | LSTM and CNN hybrid | 93.28% |

As depicted in Table 1, the prevailing emphasis in current research is on optimizing accuracy. However, the adoption of highly complex models in these studies significantly increases computational costs. For real-time analysis applications, such as web extensions, it becomes imperative to consider various factors, including data size, training and prediction times of the models

## III. DATASET SUMMARY

### A. *Data Extraction*

The process of collecting malicious and benign URLs with well-labeled formats poses a considerable challenge. Fortunately, for this research, the dataset has been sourced from Mendeley Data [11], which is a compiled dataset of phishing and legitimate websites gathered from reliable resources such as PhishTank, OpenPhish, and PhishRepo. The dataset consists of 50,000 instances of legitimate websites and 30,000 instances of phishing websites, each labeled as 0 or 1 (0 for legitimate websites, 1 for phishing websites).

To explore the possibility of reducing the computational cost, we have taken extreme measures of limiting our dataset to only 10,000 phishing and legitimate URLs. This decision is motivated by the following reasons.

- **Volatility of URLs:** URLs of this nature exhibit frequent changes in their structure, driven by attackers employing various methods such as URL masking, URL shortening, typosquatting, and malformed prefix links. Consequently, frequent model training with new datasets is essential for ensuring effective predictions.
- **Observing Model Performance:** By constraining the dataset, we aim to observe the impact on the model's performance due to a limited training set.

### B. *Feature Engineering*

The most challenging aspect of phishing detection lies in preprocessing raw URLs into independent features for model input. While some preprocessed datasets are available in certain public repositories for model training, evaluating a newly suspicious URL requires structuring it into the required format [12].

To address this challenge, we implemented a feature engineering technique that efficiently extracts the top 30 features with the highest correlations. This approach, as detailed in the referenced research [12], has been instrumental in enhancing the quality of our feature set. The URLs are labeled as phishing by the following criteria.

- **IP Address:** Presence of an IP Address in the domain.
- **Long URL:** Long URLs to hide dubious parts.
- **Tiny URL:** Usage of URL shortening services to make the URL appear legitimate.
- **"@" Symbol:** URLs containing the symbol "@".
- **Redirection by "//":** Existence of "//" within the URL path, especially after 7 indices (the default for "HTTPS" is 6).
- **Adding Prefix or Suffix:** The symbol dash (-) is rarely used in legitimate URLs.
- **Multi Sub Domain:** Indicated by the domain part containing more than one dot.
- **SSL State:** If the SSL state is not available.
- **Domain Registration Life-Period:** Domain registration period of less than a year (short lifetime period).
- **Favicon:** If the graphic image is loaded from an external source domain.
- **Using Non-Standard Port:** If the port is of a non-preferred status.
- **HTTPS:** Presence of the https token in the URL from a suspicious issuer.
- **Request URL:** If the percentage of Request URL is greater than 61%.
- **URL of Anchor:** If the percentage of Anchor tags of URL is greater than 67%.
- **Links In Tags:** If the percentage of Links in `< Meta >`, `< Script >` and `< Link>` is greater than 81%.
- **Server Form Handler:** If the SFH is different from the domain name of the webpage.
- **Submitting Information to E-mail:** Using of "mail()" or "mailto:" functions to submit user information.
- **Abnormal URL:** The hostname is not included in the URL.
- **Website Forwarding:** If the redirect pages are more than 4.
- **Status Bar Customization:** If the "onMouseOver" state changes the status bar.
- **Disabling Right Click:** If the right click is being disabled.
- **Using Pop-up Window:** If the new Popup windows contain text fields.
- **IFrame Redirection:** URL containing iframe tag.

- **Domain Age:** Having a domain of age less than a month.
- **DNS Record:** If the website has no DNS record.
- **Website Traffic:** Rank of the website is less than 100,000.
- **Page Rank:** If the PageRank is below 0.2 (The scale of page rank is from 0 – 1).
- **Google Index:** If the website is not indexed by Google.
- **Number of Links Pointing to Page:** If there's no link pointing to the webpage (98% of phishing dataset items have no links pointing to them).
- **Statistical-Reports Based Feature:** If the host belongs to top phishing IPs / Domains.

This feature engineering process helps break down the URL into distinct features, making them easy to analyze.

## IV. Dimensionality Reduction

In order to reduce the computational load of the neural network, we employ dimensionality reduction on the feature space. The dataset initially had 30 features, and the objective is to minimize dimensions while preserving correlations. Thus, we have chosen a manifold learning technique, as it retains the relationships between features more effectively than linear dimensionality reduction, even after projecting the feature space to a lower dimension.

### *A. Introduction to Manifold Learning*

Manifold learning is a subfield of topology and dimensionality reduction that focuses on capturing sensitive information within a topological space and mapping it to a lower dimension. This task proves challenging for linear dimensionality reduction algorithms like Principal Component Analysis (PCA). It can be conceptualized as an attempt to generalize linear frameworks, such as PCA, to address nonlinear structures in higher-dimensions of data.

### *B. Isomap*

Isometric Mapping (Isomap) stands out as a dimensionality reduction algorithm in manifold learning, chosen for its efficacy in unveiling the intrinsic geometric structure of high-dimensional data. Its key objective is to preserve pairwise geodesic distances between all data points, where the geodesic distance signifies the shortest path along the data manifold. This algorithm is particularly selected for the given problem due to its notable attributes. Among manifold learning algorithms, Isomap boasts one of the lowest time complexities while maintaining high accuracy in mapping, making it a compelling choice for efficient and accurate dimensionality reduction in this context.

**Algorithm:**
**Input***:* *Data vectors* $x_1, x_2, \ldots, x_n \in \mathbb{R}^D$ *with parameters* $k$ *(nearest neighbors) and* $m$ *(embedding dimensions.*
**Output:** *Lower dimensional embedding vectors* $y_1, y_2, \ldots, y_n \in \mathbb{R}^d$ *with* $d < D$*.*
**Steps:**

- Nearest neighbor search using ball tree algorithm.
- Shortest-path graph search Dijkstra's algorithm.
- Partial eigenvalue decomposition.

**Key Components:**

- *N*: number of training data points
- *D*: input dimension
- $k$: number of nearest neighbors
- *d*: output dimension

*1) Ball Tree:* The Ball Tree algorithm is a spatial data structure designed for efficient nearest neighbor searches in multidimensional spaces. It was introduced as a solution to expedite distance-based queries, particularly k-nearest neighbor searches. The fundamental concept behind the Ball Tree involves recursively partitioning the dataset into bounding hyperspheres, commonly referred to as "balls," in a hierarchical manner. At each level of the tree, a representative point (centroid) is chosen to define the center of the enclosing ball, thereby streamlining the search process. The time complexity of the algorithm is $O[D\,log(k)N\,log(N)]$ [13].

*2) Dijkstra's Algorithm:* It is an integral component of Isomap, is pivotal for determining the shortest path distances between data points. In the context of Isomap, it facilitates the construction of a geodesic distance matrix. By treating data points as nodes in a graph and utilizing pairwise distances as edges, Dijkstra's algorithm efficiently computes geodesic distances through an iterative process. This contributes to an accurate representation of the underlying manifold structure, enhancing Isomap's effectiveness in preserving intrinsic relationships during dimensionality reduction. The time complexity of Dijkstra's Algorithm within Isomap is $O[N^2(k + log(N))]$ [13].

*3) Partial eigenvalue decomposition:* It plays a pivotal role in Isomap, involving the symmetric Laplace matrix derived from the neighborhood graph. Unlike computing all eigenvalues, PED strategically focuses on calculating only the smallest ones, corresponding to the low-dimensional representation. This decision significantly improves computational efficiency, a crucial aspect of the algorithm. The eigenvectors associated with these smallest eigenvalues collectively form the low-dimensional representation of the data. Stacking these eigenvectors creates the final embedding matrix, capturing the intrinsic geometry of the manifold. PED contributes to the overall computational effectiveness of Isomap in preserving essential structural features, with an approximate complexity of $O[dN^2]$ [13].

### *C. Results*

By applying the Isomap algorithm to our dataset, we have successfully reduced the feature space from 30 to just 5 features. It is worth noting that attempts to further decrease the feature space below 5 results in a substantial loss of correlation. Consequently, we have opted to limit the reduction to this dimensionality. The modeled neural network will now operate on this reduced set of features, leading to a significant decrease in computational costs. The overall time complexity of the Isomap algorithm, considering the dimensionality reduction, is expressed as

$O[D\,log(k)N\,log(N)] + O[N^2(k + log(N))] + O[dN^2]$. This reduction in the feature space contributes to enhanced computational efficiency, aligning with the overarching goal of minimizing computational costs in neural network modeling.

## V. Proposed Model

In this research, we address the pressing challenge of achieving computational efficiency in neural network modeling for binary classification tasks. Acknowledging the significance of minimizing computational costs, we propose a simple architecture specifically constructed for scenarios characterized by limited computational resources. The neural network is composed of three layers that are designed to achieve an equilibrium between model complexity and its predictive capacity.

### A. Activation Function

*1) Sigmoid Function:* The logistic function, commonly known as the sigmoid function, holds particular utility in binary classification tasks, primarily owing to its capacity to map real-valued inputs to a bounded range of 0 to 1. This characteristic is pivotal for interpreting the function's output as a probabilistic score, where values close to 1 denote phishing, and those near 0 denote legitimate URLs.

The time complexity of the sigmoid function itself is generally considered to be low. The sigmoid function is a simple mathematical expression [14].

$$\sigma(x) = \frac{1}{1 + e^{-z}} \tag{1}$$

Where $e$ is the base of the natural logarithm. The computation involves exponentiation and a few computationally inexpensive arithmetic operations Thus, it is chosen for the output node of the neural network.

For backward propagation, we compute the derivative of the sigmoid function, given as:

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big) \tag{2}$$

*2) ReLU Function:* The Rectified Linear Unit (ReLU) activation function is computationally efficient and has low time complexity. The ReLU function is defined as [14]:

$$f(x) = max(0, x) \tag{3}$$

It implies that it provides the input value if it is positive and zero otherwise.

The time complexity of evaluating the ReLU function for a single input is very low, involving a simple comparison and a maximum operation. These operations can be efficiently implemented in our neural network for Layer 1 and Layer 2.

For backward propagation, we compute the derivative of the ReLU function which is given as:

$$relu'(x) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases} \tag{4}$$

### B. Neural Network Architecture

The initial layer consists of 25 nodes employing Rectified Linear Unit (ReLU) activation, succeeded by a subsequent layer featuring 15 nodes, thereby enhancing the model's capability to capture nonlinearities in feature representation. The output layer, encompassing a single node activated by the sigmoid function, facilitates binary classification. The objective is to enhance model performance while mitigating computational burdens, accomplished by utilizing a dataset with dimensions of (10000, 5) for input features 'X' and (10000, 1) for labels 'y'. To realize this goal, we employ the binary cross-entropy log loss function and the Adam optimizer. Through this tailored neural network configuration, we aim to underscore the efficacy of a minimalist architecture in achieving a favorable balance between computational efficiency and predictive accuracy, thereby laying the foundation for resource-efficient solutions in real-world applications.

TABLE II. Neural Network Layers

| | ***Shape of W*** | ***Shape of b*** | ***Activation*** | ***Shape of Activation*** |
|---|---|---|---|---|
| Layer 1 | (25, 5) | (25, 1) | $z^{[1]} = W^{[1]}X + b^{[1]}$ | (25, 10000) |
| Layer 2 | (15,25) | (15, 1) | $z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$ | (15, 10000) |
| Layer 3 | (1, 15) | (1, 1) | $z^{[3]} = W^{[3]}A^{[2]} + b^{[3]}$ | (1, 10000) |

**Key Components:**

$W$ – Weight
$b$ – bias
$Z$ - Weighted sum
$A$ - Activation layer
Superscripts square brackets – Layers
Superscript parenthesis – Training examples

The Loss function, denoted by $\mathcal{L}$, to compute the error for one training sample is given below [14]:

$$\mathcal{L}\big(a^{(i)}, y^{(i)}\big) = -y^{(i)} \log\big(a^{(i)}\big) - \big(1 - y^{(i)}\big) \log\big(1 - a^{(i)}\big) \tag{5}$$

The Cost function $\mathcal{J}$ is then computed by summing over all training examples [14]:

$$\mathcal{J} = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\big(a^{(i)}, y^{(i)}\big) \tag{6}$$

### C. Derivatives for Backpropagation

For computing $dZ^{[l]}$ we can use (2) or (4) depending upon the activation. To perform backpropagation, it is important to compute the following derivatives [14].

$$dW^{[l]} = \frac{\partial \mathcal{J}}{\partial W^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l-1]T} \tag{7}$$

$$db^{[l]} = \frac{\partial \mathcal{J}}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)} \tag{8}$$

$$dA^{[l-1]} = \frac{\partial \mathcal{L}}{\partial A^{[l-1]}} = W^{[l]T} dZ^{[l]} \tag{9}$$

## D. Optimization

A simple optimization method in machine learning is gradient descent. When taking gradient steps with respect to all $m$ examples on each step, it is also called Batch Gradient Descent. The gradient descent rule is, for $l = 1, \ldots\ldots L$: [15]

$$W^{[l]} = W^{[l]} - \alpha\, dW^{[l]} \tag{10}$$

$$b^{[l]} = b^{[l]} - \alpha\, db^{[l]} \tag{11}$$

Momentum takes into account the past gradients to smooth out the update. We store the 'direction' of the previous gradients in the velocity variable $v$.

$$v_{dW^{[l]}} = \beta v_{dW^{[l]}} + (1-\beta) dW^{[l]} \tag{12}$$

$$W^{[l]} = W^{[l]} - \alpha v_{dW^{[l]}} \tag{13}$$

where $l$ is the number of layers, $\beta$ is the momentum and $\alpha$ is the learning rate.

Adam optimization: It calculates an exponentially weighted average of past gradients, storing it in variables $v$ (before bias correction) and $v^{corrected}$ (with bias correction). It calculates an exponentially weighted average of the squares of the past gradients, storing it in variables $s$ (before bias correction) and $s^{corrected}$ (with bias correction). Then it updates parameters in a direction based on combining information from previous steps.

The update rule is, for $l = 1, \ldots\ldots L$: [15]

$$v_{dW^{[l]}} = \beta_1 v_{dW^{[l]}} + (1-\beta_1)\frac{\partial \mathcal{J}}{\partial W^{[l]}} \tag{14}$$

$$v^{corrected}_{dW^{[l]}} = \frac{v_{dW^{[l]}}}{1-(\beta_1)^t} \tag{15}$$

$$s^{corrected}_{dW^{[l]}} = \frac{s_{dW^{[l]}}}{1-(\beta_2)^t} \tag{17}$$

$$W^{[l]} = W^{[l]} - \alpha \frac{v^{corrected}_{dW^{[l]}}}{\sqrt{s^{corrected}_{dW^{[l]}}} + \varepsilon} \tag{18}$$

Where $\beta_1$and $\beta_2$ are hyperparameters that control the two exponentially weighted averages. and $\varepsilon$ is a very small number to avoid dividing by zero. $t$ counts the number of steps taken by Adam optimizer.

## VI. Computational Cost

In the realm of machine learning algorithms, accurately determining time complexities can be challenging. However, in this specific section, our aim is to provide the most precise approximations possible.

Neural networks commonly process data in the form of tensors. While frameworks such as NumPy, Pandas, and TensorFlow efficiently manage tensor computations through parallel and distributed computing, we will base our time complexity analysis on the simplest multiplication algorithm for the sake of easier understanding approximation.

### A. Forward Propagation

The proposed model has 3 layers, to compute generalized time complexity let us assign the following:

Let $i, j, k, l$ be the number of nodes in each layer.
$m$ – number training set

Let the weights be as $W_{ji}$, $W_{kj}$ and $W_{lk}$

Propagation from layer $i \rightarrow j$

$$Z_{jm} = W_{ji} * X_{im} \tag{19}$$

This operation has a complexity of $\mathcal{O}(i * j * m)$
Applying activation:

$$A_{jm} = f\left(Z_{jm}\right) \tag{20}$$

This has a complexity of $\mathcal{O}(j * m)$, because both sigmoid and ReLU are element-wise operations.

$$\begin{aligned} \mathcal{O}(j * i * m + j * m) &= \mathcal{O}\big(j * m * (i+1)\big) \\ &= \mathcal{O}(j * i * m) \end{aligned} \tag{21}$$

For the layer $j \rightarrow k$:

$$\mathcal{O}(k * j * m) \tag{22}$$

and $k \rightarrow l$:

$$\mathcal{O}(l * k * m) \tag{23}$$

From (21), (22) & (23) the time complexity for forward propagation will be:

$$\mathcal{O}(j * i * m + k * j * m + l * k * m)$$

$$\mathcal{O}\big(m * (ij + jk + kl)\big) \tag{24}$$

### B. Backward Propagation

The time complexity of back-propagation algorithm is derived as follows. Starting from the output layer $l \rightarrow k$, then we compute the loss, $E_{lm}$ a matrix of signals for nodes at layer $l$.

$$E_{lm} = f'(Z_{lm}) \odot (A_{lm} - y_{lm}) \tag{25}$$

Where $\odot$ means element-wise multiplication

Note: $E_{lm}$ has $l$ rows and $m$ columns, which simply means each column is the error signal for training example $m$.

We then compute delta weights, $D_{lk} \in R^{l \times k}$ (between layer $l$ and layer $k$)
.

$$D_{lk} = E_{lm} * A_{mk} \tag{26}$$

Where $A_{mk}$ is the transpose of $A_{km}$

We then adjust the weights

$$W_{lk} = W_{lk} - D_{lk} \tag{27}$$

For $l \rightarrow k$, the time complexity is

$$\mathcal{O}(lm + lm + lmk + lk) = \mathcal{O}(l * m * k) \tag{28}$$

For $k \rightarrow j$. We have the following

$$E_{km} = f'(Z_{km}) \odot (W_{kl} * E_{lm}) \tag{29}$$

$$D_{kj} = E_{km} * A_{mj} \quad (30)$$

$$W_{kj} = W_{kj} - D_{kj} \quad (31)$$

From (29), (30) & (31), The time complexity is

$$O(km + klm + kmj + kj) = O\big(k * m(l + j)\big) \quad (32)$$

And finally, for $j \to i$, we have

$$O\big(j * m(k + i)\big) \quad (33)$$

In total, we have $O\big(lmk + mk(l + j) + mj(k + i)\big)$ which can be simplified as:

$$O\big(m * (lk + kj + ji)\big) \quad (34)$$

which is the same as the forward propagation (24). Since they are the same, the total time complexity for one epoch will be

$$O\big(m * (ij + jk + kl)\big).$$

This is then multiplied by the number of iterations (epochs). So, we have

$$O\big(n * m * (ij + jk + kl)\big) \quad (35)$$

We made the simplifying assumption of employing the most basic matrix multiplication, characterized by cubic time complexity. This complexity remains consistent across various optimization techniques, including mini-batch gradient descent, stochastic optimization, and momentum optimization such as the Adam optimizer.

## VII. Experimental Results

TABLE III. Model Performance

| *Epoch* | *Precision* | *Recall* | *Accuracy* |
|---|---|---|---|
| 1 | 76.21% | 80.74% | 75.23% |
| 10 | 87.16% | 86.75% | 85.50% |
| 20 | 88.48% | 87.43% | 86.66% |
| 30 | 88.96% | 87.66% | 87.06% |

The accuracy of the predictions from the three-layer neural network closely rivals that of the high-complexity models presented in Table 1. The proposed algorithm achieves an impressive 87% accuracy after just 30 epochs, notably compared to the 93.28% accuracy achieved by the LSTM and CNN hybrid and the 92% accuracy from Ensemble methods. Consequently, the conclusion drawn is that the proposed model substantially diminishes computational costs, incurring only a 5% accuracy reduction when compared to the top-performing algorithms.

## VIII. Conclusion

In the context of real-time analysis, such as that of a web extension URL filter, the emphasis on computational efficiency becomes paramount due to diverse factors. The dynamic nature of URL features necessitates frequent model training with the integration of new features. The proposed model stands out for its prowess in minimizing computational costs, surpassing leading phishing detection models, albeit with a marginal compromise in accuracy. Notably, the disparity between our proposed model and top-performing models ranges only from 5 to 8%. Future research endeavors will concentrate on elevating accuracy while preserving a low computational threshold.

## References

[1] CISCO, "Cyber security threat trends: phishing," 2021. [Online]. Available: https://umbrella.cisco.com/info/2021-cyber-security-threat-trends-phishing-crypto-top-the-list.

[2] M. a. I. Y. a. J. A. Khonji, "Phishing Detection: A Literature Survey," *IEEE Communications Surveys & amp Tutorials,* pp. 1-31, 2013.

[3] D. L. a. P. S. a. P. Glancy, "Taking the Bait: A Systems Analysis of Phishing Attacks," *Procedia Manufacturing,* vol. 3, pp. 1109-1116, 2015.

[4] IBM, "Cost of a data breach report," 2022-2023. [Online]. Available: https://www.ibm.com/reports/data-breach.

[5] APWG, "Phishing activity trends report," 2022. [Online]. Available: https://apwg.org/trendsreports/.

[6] M. M. D. a. M. I. Vahid Shahrivari, "Phishing Detection Using Machine Learning Techniques," *arXiv,* 2020.

[7] Z. a. A. R. a. A.-M. J. a. H. Q. E. U. a. S. K. a. F. M. H. Alshingiti, "A Deep Learning-Based Phishing Detection System Using CNN, LSTM, and LSTM-CNN," *Electronics,* vol. 12, p. 1, 2023.

[8] H. S. a. I. R. Katherine Haynes, "Lightweight URL-based phishing detection using natural language processing transformers for mobile devices," *Procedia Computer Science,* vol. 191, pp. 127-134, 2021.

[9] A. a. A. A.-H. Q. a. A. A. a. a. t. A. Odeh, "Comparative Study of CatBoost, XGBoost, and LightGBM for Enhanced URL Phishing Detection: A Performance Assessment," *Journal of Internet Services and Information Security,* vol. 13, pp. 1-11, 12 2023.

[10] A. A.-A. a. S. Al-Ahmadi, "Robust URL Phishing Detection Based on Deep Learning," *KSII Transactions on Internet and Information Systems,* vol. 14, pp. 2752-2768, 2020.

[11] S. Ariyadasa, S. Fernando and S. Fernando, "Phishing Websites Dataset," University of Moratuwa, Uva Wellassa University, 17 11 2021. [Online]. Available: https://data.mendeley.com/datasets/n96ncsr5g4/1.

[12] R. M. F. T. a. L. M. Mohammad, "Phishing websites features," School of Computing and Engineering, University of Huddersfield, 2015.

[13] J. B. T. a. V. d. S. a. J. C. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction," *Science,* vol. 290, pp. 2319-2323, 2000.

[14] C. M. Bishop, Pattern Recognition and Machine Learning, 1 ed., Cambridge: Springer New York, NY, 2006, p. 778.

[15] D. P. K. a. J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980,* 2017.