Computers and Electrical Engineering 119 (2024) 109494

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Context-aware embeddings for robust multiclass fraudulent URL detection in online social platforms

Sara Afzal [a], Muhammad Asim [a], Mirza Omer Beg [a], Thar Baker [b], Ali Ismail Awad [c,d,*], Nouman Shamim [e]

[a] School of Computing, National University of Computer and Emerging Sciences, Islamabad 44000, Pakistan
[b] School of Architecture, Technology and Engineering, University of Brighton, Brighton BN2 4GJ, UK
[c] College of Information Technology, United Arab Emirates University, Al Ain, P.O. Box 15551, United Arab Emirates
[d] Big Data Analytics Center, United Arab Emirates University, Al Ain, P.O. Box 15551, United Arab Emirates
[e] Pakistan Institute of Engineering and Applied Sciences (PIEAS), Islamabad, Pakistan

ARTICLE INFO

Keywords:
Artificial neural networks
BERT embeddings
URL classification
Securing online social networks

ABSTRACT

The current ubiquity of online social networks (OSNs) cannot be overstated, and they have over 4.8 billion users worldwide. These platforms have become integrated into modern life, representing an important means of communication and information sharing. However, this widespread popularity has also drawn the attention of cybercriminals, who seek to exploit OSNs using deceptive Uniform Resource Locators (URLs) as their weapons of choice. Conventional URL-classification methods, which rely on post-access features or static analysis, face significant limitations; they struggle to keep pace with the ever-evolving tactics of cybercriminals, and they often lack the granularity required for precise URL categorization. The methodology proposed herein takes a different path, leveraging the power of an artificial neural network (ANN) in tandem with Bidirectional Encoder Representations from Transformers (BERT) to extract contextual embeddings from URLs. By combining the cutting-edge capabilities of ANNs and BERT, we introduce an efficient approach to safeguarding OSN users from the insidious threats lurking behind deceptive URLs by classifying them into five distinct categories: benign, defacement, phishing, malware, and spam. The proposed approach was found to achieve an impressive accuracy rate of 98.0%, surpassing the previous best of 97.92%. This technique thus has the potential to serve as a crucial defense mechanism for the billions of individuals who rely on OSNs for their social and informational needs.

1. Introduction

The Internet is an indispensable cornerstone of contemporary society; it facilitates communication and the exchange of knowledge and ideas, serving as a vital resource for billions worldwide, spanning both essential and discretionary needs. Notably, the digital landscape is dominated by online social networks (OSNs) – sites such as Facebook, Twitter, and Instagram – which have successfully harnessed the attention of a very large proportion of online users. With the increasing number of individuals actively engaged in OSNs, people's susceptibility to cyberattacks [1] by malicious actors is significantly heightened. This ominous trend is demonstrated by an abundance of phishing and malware incidents being detected, shedding light on the malevolent objectives of cybercriminals,

* Corresponding author at: College of Information Technology, United Arab Emirates University, Al Ain, P.O. Box 15551, United Arab Emirates.
E-mail addresses: sara.afzal@nu.edu.pk (S. Afzal), muhammad.asim@nu.edu.pk (M. Asim), omer.beg@nu.edu.pk (M.O. Beg), T.Shamsa@brighton.ac.uk (T. Baker), ali.awad@uaeu.ac.ae (A.I. Awad), nauman@pieas.edu.pk (N. Shamim).

https://doi.org/10.1016/j.compeleceng.2024.109494
Received 18 December 2023; Received in revised form 14 June 2024; Accepted 16 July 2024
Available online 30 July 2024

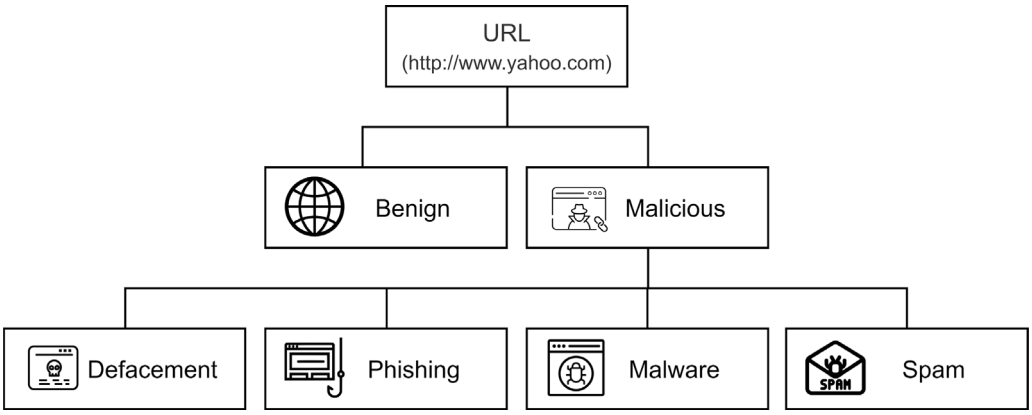

Fig. 1. Multiclass classification of URLs.

who seek to exploit any potential vulnerabilities, aiming to gain unauthorized system access, steal credentials for financial gain, or advance nefarious agendas [2]. Nevertheless, the realm of information security offers opportunities to proactively counteract such attacks by increasing both user and data security.

Even with appropriate vigilance, attack prevention is an uphill battle. The adept mimicking behavior of fake websites can make it extremely hard for some people to assess whether they are genuine. Attackers have been using false Uniform Resource Locators (URLs) on several websites, deceiving users into clicking on them while believing they are accessing legitimate URLs. It can be challenging to tell a benign URL from a malicious one because their attributes can appear genuine. Blacklisting, which entails adding known harmful URLs to a list, is one of the most widely used techniques in this area; however, this method does not stop people from accessing or reading a dangerous URL not on that list. As such, this strategy should not be trusted because the risk persists [3–5].

In this field, researchers have concentrated on classifying harmful and genuine URLs using cutting-edge approaches to shield people and systems from potential harm. To persuade users to visit a malicious URL, an attacker will commonly replicate the qualities of a benign URL to fool them; this can result in financial losses, the theft of private information, drive-by download attacks, and other losses. The number of such attacks is continuously increasing with the increasing number of OSN users; at the same time, new solutions are being developed as researchers discover new approaches and tools. Because much information is exposed to the public Internet, attackers can potentially obtain large amounts sensitive data. For this reason, they continue to create ever more novel attack methods of which no user is yet aware [6]. Attackers will often focus on a particular organization, system, or user if there is no structure in place to stop them.

Federated learning [7] is a method that offers advantages for classification of URLs as either harmless or malicious; it can also help to maintain user privacy by training models on separate devices and aggregating data without central collection. This enables the use of a wider variety of training data, improving the model's understanding of different URL formats. Nonetheless, federated learning also has drawbacks, such as the need for back-and-forth communication between the main server and devices, which can cause lag, and the possibility of data differences from multiple devices complicating the learning process. Security concerns also need to be considered, as malicious actors may tamper with the model itself or leak data.

As a result of the problems outlined above, it is essential to create systems that can defend against novel attacks. This requires studies to be conducted in this area to develop methods for recognizing dangerous URLs and understanding their categories before they are clicked on. It is important to take action as soon as a malicious URL is identified. For instance, if a URL is flagged as pointing to a defaced page, it has to be fixed promptly [3,8]; nevertheless, it can be responded to later if identified as a malicious URL.

Fig. 1 shows the ways in which a URL can be classified as either benign or malicious. Malicious URLs can be further classified into four categories: defacement, phishing, malware, and spam. Failing to categorize malicious URLs into specific types such as these can lead to a range of issues in cybersecurity: it can result in an ineffective threat response, missed detection of specific threats, higher rates of false positives and negatives, inefficient allocation of security resources, compliance and reporting challenges, or reduced situational awareness. Proper categorization is indeed essential for targeted mitigation, accurate threat identification, regulatory compliance, and maintaining an effective security posture.

Because of the importance of dealing with all types of malicious URLs, in this study, we sought to design a technique to overcome issues in existing approaches using Bidirectional Encoder Representations from Transformers (BERT) embeddings and artificial neural networks (ANNs). In our approach, an ANN interacts with BERT by using it as a feature extractor and integrating it into a classification pipeline. BERT's contextualized embeddings provide powerful representations of the input text, which are then fed into the ANN to learn and predict the class labels for text-classification tasks.

Table 1
List of abbreviations.

Abbreviation	Full form
ANN	Artificial neural network
AUC	Area under the receiver operating characteristic curve
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional neural network
DNN	Deep neural network
FNR	False-negative rate
FPR	False-positive rate
GRU	Gated recurrent unit
LSTM	Long short-term memory
NLP	Natural language processing
OSN	Online social network
ReLU	Rectified linear unit
SVM	Support vector machine
TF-IDF	Term frequency–inverse document frequency
TPR	True-positive rate
URL	Uniform Resource Locator

1.1. Motivations and contributions

A recent surge in malicious URLs is posing a substantial cybersecurity challenge, given their evolving complexity and diverse nature. Although simply detecting these threats remains a critical endeavor, the significance of classifying them cannot be overstated. Our research is underpinned by the recognition that beyond mere detection, precise categorization of malicious URLs is imperative. By categorizing threats into distinct types such as spam, phishing, malware, and defacement, we can gain deeper insights into their specific characteristics and malicious intent. This approach allows cybersecurity responses to be more effectively tailored, enhancing not only the accuracy of threat identification but also the precision and efficiency of mitigation efforts. As such, this paper seeks to provide a notable contribution to this vital area, offering valuable insights and methodologies to fortify defenses against the continually evolving landscape of online threats.

Herein, we propose a new multiclass URL-classification method for detecting malicious URLs. This combines the BERT approach with an ANN for higher accuracy. The contributions of this work can be summarized in the following points:

- A novel application of BERT embeddings is introduced to generate contextualized representations of URLs, enhancing their classification accuracy.
- The traditional binary-classification approach is extended using an ANN, enabling multiclass classification of URLs into five distinct categories: benign, defacement, phishing, malware, and spam.
- A comprehensive dataset comprising diverse URL types is curated and used to demonstrate the performance of the proposed BERT-based model across various categories to ensure robust evaluation.
- An accuracy of 98.0% is achieved, outperforming existing approaches through extensive experimentation and demonstrating its potential for practical online security applications.

1.2. Paper structure

The remainder of this paper is structured as follows. In Section 2, the related work and identified research gap are presented. The proposed multiclass URL-classification technique is described in Section 3. Section 4 covers the experimental setup, along with the results obtained and a comparative analysis. Finally, Section 5 concludes the work and highlights potential future research directions.

2. Related work

This section presents an analysis of the state of the art of malicious URL detection and classification. It is divided into two broad categories: machine-learning models and deep-learning models. Table 1 lists the abbreviations used in this paper.

2.1. URL classification using machine-learning models

Bahnsen et al. [9] tackled the challenge of identifying harmful URLs in imbalanced datasets using a cost-sensitive (CS) XGBoost approach. They emphasized the importance of addressing data imbalance, especially in critical applications such as fraud detection. Their study compared CS-XGBoost with XGBoost and SMOTE+XGBoost using a substantial dataset of 6 million URLs and 28 diverse features. These features encompass domain attributes, geographic data, WHOIS information, and word-based indicators of suspicion. The authors discussed the limitations of common solutions to data imbalance, and they employed t-distributed stochastic neighbor

embedding for dimensionality reduction. Their evaluation – which features ten-fold cross-validation and metrics such as the G-mean, the area under the receiver operating characteristic curve (AUC), and sensitivity – underscored the superior performance of CS-XGBoost in comparison to other methods.

Johnson et al. [10] conducted two experiments. The first of these sought to detect and categorize URLs as either benign or malicious; they found that the random-forest algorithm achieved the highest binary-classification accuracy at 98.68%, while the fast.ai and Keras-TensorFlow deep-learning models achieved 96.88% and 97.13%, respectively. In the second experiment, malicious URLs were categorized into defacement, spam, malware, and phishing; here, the random-forest, fast.ai, and Keras-TensorFlow approaches achieved multiclass classification accuracies of 96.26%, 96.77%, and 91.45%, respectively. The study employed 78 features and the ISCX-URL-2016 dataset, and it used feature-ranking methodologies to identify the top-ten features yielding the highest accuracy. The evaluation metrics employed to support their findings included performance accuracy, confusion matrices, and prediction timeframes.

Mamun et al. [11] implemented two key strategies: detecting and classifying malicious URLs and investigating the impact of obfuscation techniques on such URLs. They analyzed four types of malicious URL (spam, malware, phishing, and defacement) using 79 lexical features and four obfuscation-related features. With an initial dataset of 114,400 URL entries, 110,000 were used for experimentation. Three machine-learning techniques (k-nearest neighbors (KNN), C4.5, and random forest) and two feature-selection algorithms (CfsSubsetEval and InfoGain) were employed. It was found that random forest outperformed the other methods, achieving accuracy rates of 99% and 97% for single-class and multiclass datasets, respectively.

Kumi et al. [12] introduced a novel approach using association-based rules to detect both malicious and benign URLs. They argued that conventional blacklisting methods are insufficient for detecting emerging malicious URLs. Their method employs associative classification to discover associations among patterns and reveal hidden insights that are often missed by other models. They addressed the limitations of existing techniques that focus on single attack types. Their dataset was constructed from Alexa's top-500 sites for benign URLs and sources including OpenPhish, VX Vault, and URLhaus for malicious URLs, totaling 1200 URLs (700 malicious and 500 benign). Using Python and R, they performed ten-fold cross-validation and reported evaluation metrics including precision (91.30%), recall (97.67%), AUC (96.2%), and false-positive area (8.6%), demonstrating the effectiveness of their approach.

Telo et al. [13] used supervised machine-learning techniques such as random forest, decision tree, Gaussian naive Bayes, KNN, stochastic gradient descent, extra trees, and AdaBoost to classify URLs into four categories; however, they used static features such as the count of symbols such as "#" and "@" in a URL. They achieved an overall accuracy of only 91.4%, which means that there is significant room for improvement.

Chiramdasu et al. [14] focused on malicious-URL detection using logistic regression. They categorized malicious URLs and performed feature engineering, employing a dataset comprising 32,000 URLs from various sources. Their framework extracts 12 features, including host-based, domain-based, and lexical features. This system was implemented in Python, and they used five-fold cross-validation. Their evaluation metrics included accuracy, precision, recall, F-measure, true-positive rate (TPR), and false-positive rate (FPR). When compared with the support vector machine (SVM), linear discriminant analysis, and KNN approaches, their technique outperformed all models, achieving 91% accuracy, 90% F-measure, 85% recall, 62% precision, 78% FPR, and 80% TPR.

Rupa et al. [15] addressed the growing threat of malicious URLs by designing a machine-learning framework to detect fraudulent URLs, aiming to protect users and organizations. Their model aims to classify malicious URL types, including phishing, spam, and malware, and it uses feature extraction followed by random-forest classification. The feature information extracted encompasses lexical, URL, and malicious keyword features. They gathered datasets from various sources, including search engines, customer-care centers, PhishTank, Kaggle, and online shopping domains. Their framework consists of two modules: verification and extraction. The model was found to achieve an impressive accuracy of 98.96% on open datasets, offering robust protection against malicious URLs.

Wejinya et al. [16] highlighted the vulnerability of users – especially those with limited knowledge of cyberattacks – to malicious URLs. They emphasized the shortcomings of traditional methods such as blacklisting and heuristics for the timely detection of new harmful URLs. Their proposed model involves lexical, host-based, and content-based feature extraction, and classification. They used a dataset from the UCI Machine Learning Repository comprising 11,055 URLs and 30 features, reducing them to 15 final features for machine learning. They employed SVM, naive Bayes, and logistic regression for classification, achieving impressive accuracies of 100%, 98%, and 96%, respectively. Their evaluation criteria included accuracy, precision, recall, and F-measure.

Aljabri et al. [17] used a dataset consisting of a total of 66,506 URLs to conduct a binary classification of URLs. They also used feature-selection techniques as well as applying machine-learning and deep-learning models to compare their results with existing approaches. They achieved an accuracy of 96%; however, the limitations of their work include the use of static features such as a count of the parts of URLs, the lengths of different paths, and geolocation; these features do not provide the contextual meaning of a URL and are unable to provide good results for unseen data.

Chawla et al. [18] addressed the need to protect the public from phishing scams. They used the random-forest, decision-tree, logistic-regression, KNN, ANN approaches, along with the Max Vote Classifier of random forest, to train their data. Random forest was found to achieve the highest accuracy of 97.73%; however, their use of features was not very reliable. This is because they also used static features, the pattern of which is continuously changing with new attacks and new tactics of attackers in the creation of malicious URLs.

2.2. URL classification using deep-learning models

He et al. [19] focused on phishing-URL classification using long short-term memory (LSTM) neural networks and a random forest approach. They extracted features from URLs automatically by learning representations from character sequences. In their system, a 128-dimension embedding translates each input character. They addressed the limitations of recurrent neural networks in capturing long-range correlations and employed an LSTM to bridge elements even when they are separated by 1000 steps. The translated URLs, treated as a 150-step sequences, are fed into an LSTM layer for classification using a sigmoid neuron. Their dataset comprised 2 million URLs from Common Crawl and PhishTank, and they applied three-fold cross-validation with 90% training data. Feature-ranking techniques were used to identify the most important among 14 features. They achieved an accuracy of 98.7% with LSTM and 93.5% with random forest.

Le et al. [20] criticized traditional methods for their limitations in capturing URL semantics, heavy feature engineering, and poor generalization. They introduced a novel approach: nonlinear URL embeddings for malicious-URL detection; however, these embeddings face challenges, such as difficulty handling new words and high memory usage. To address these issues, they combined character and word embeddings. Their dataset comprised 15 million URLs from VirusTotal, with 5 million for training and 10 million for testing. Their model achieved an impressive accuracy of 99.29%, demonstrating the system's effectiveness in overcoming the limitations of traditional systems.

Rao et al. [21] conducted five experiments to detect fraudulent URLs using a dataset containing HTML source markup from various websites. This dataset comprised entries from 5438 phishing websites and 5076 authentic website. The majority of these data were allocated for training, with the remainder used for testing. They employed term frequency–inverse document frequency (TF-IDF) and word2vec to create vectors for model input. The experiments involved ordinary text, domain-specific content, a combination of plain-text word embedding with domain-specific data, and a comparison with previous techniques. Their multimodal approach outperformed other approaches with a remarkable accuracy of 99.34%, a Matthews correlation coefficient of 98.68%, a TPR of 99.59%, and a true-negative rate of 99.07%. However, they identified limitations in their study, including model failure with image-based text and memory-efficiency challenges due to document-size averaging.

Do et al. [22] conducted a comprehensive comparison of different deep-learning techniques, emphasizing their unique contributions in terms of parameter settings, hyperparameter optimization, and performance metrics. They evaluated four distinct models: a convolutional neural network (CNN), a deep neural network (DNN), a gated recurrent unit (GRU), and an LSTM network, using a dataset from the UCI Machine Learning Repository containing 11,055 URLs (4898 phishing and 6157 legitimate). They allocated 80% for training and 20% for testing. With 30 features and neurons in the input layer, they used Google Colaboratory, Python, and TensorFlow for training, implementing batch normalization and dropout to prevent overfitting. Their results showed accuracy rates of 97.29% (DNN), 96.56% (CNN), 97.20% (LSTM), and 96.70% (GRU). Future plans include testing on a larger, unbalanced dataset for greater real-world relevance.

Saxe et al. [23] introduced the eXpose neural-network model, which is designed to analyze generic and raw short character strings as inputs to extract features and determine the malicious or benign nature of URLs using character embeddings and a CNN. Their method eliminates the need for manual feature engineering, resulting in a 5%–10% higher detection rate. They implemented the model in Python 2.7 using Keras v1.1, with a dataset collected from VirusTotal comprising 19,067,879 URLs with imbalanced malicious and benign entries. To address this data imbalance, they use batch streaming. Two baseline models, a standard n-gram extractor and manual feature extraction with a 1024-dimensional feature vector, were employed. The TPR and AUC values were used as evaluation metrics, yielding impressive results, with an AUC of 0.993 and TPR values of 0.77, 0.84, and 0.92 at FPRs of 10^{-4}, 10^{-3}, and 10^{-2}, respectively.

Prabakaran et al. [24] discussed the reliance of traditional defenses against phishing techniques on blacklisting methods and the resulting difficulty they have keeping up with the rapid creation of new malicious sites. In consequence, they proposed a deep-learning approach combining variational autoencoders and DNNs. The study tested the model on 100,000 publicly available records, achieving an accuracy of 97.45% for binary classification. However, the effectiveness of the approach can be lower when faced with previously unseen phishing URLs, as attackers are using new tactics to evolve their phishing techniques.

Patgiri et al. [25] used a Bloom filter, which is a type of data structure for data retrieval and membership testing. This functions like a digital sieve to test different ways of processing text to improve performance. After this, they used an evolutionary CNN to find malicious websites. However, using a Bloom filter has some limitations, such as its large memory usage, which can make this technique inefficient for large datasets. Furthermore, the performance of a Bloom filter depends on the quality of the hash function used; a poorly chosen hash function can lead to an increased FPR or decreased efficiency.

2.3. Research-gap analysis

To summarize our findings, we can say that online social platforms have become a popular target for cybercriminals for a variety of reasons; these include stealing credentials for money, putting malware in a system to gain access to personal data, and demanding money using ransomware. These actions can be carried out through malicious URLs, so a user may be unable to tell if something is wrong with their system. Previous research [3,14,20,26] has covered a substantial amount of work in this field, including the binary classification of malicious and benign URLs. However, very few studies have emphasized working with multiclass classification, and those that have considered this have mainly examined two to three additional categories. Furthermore, the majority of previous works have made use of machine-activity features, which can be collected by clicking a URL. This can affect the system or the user before a URL is detected and blocked. Other works have extracted lexical [5], domain [9], and host-based [11] features that do

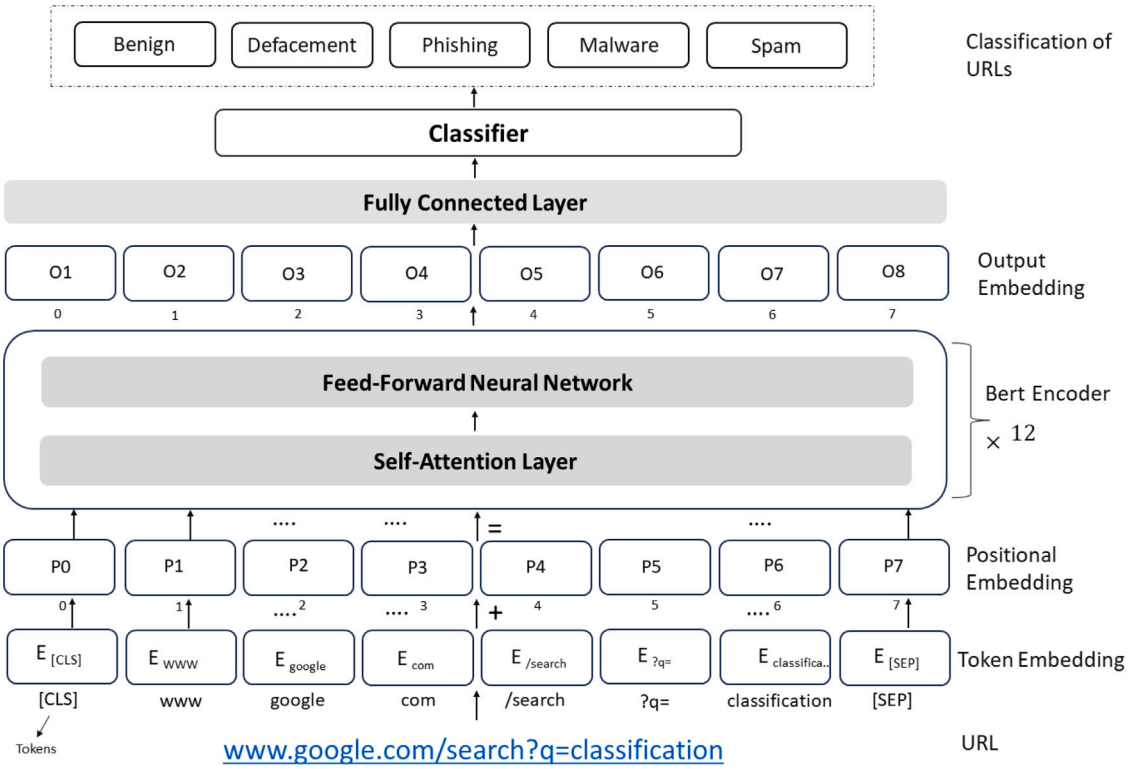

Fig. 2. Overview of the method employed for classification of URLs into five categories. The process begins with the input URL, which undergoes tokenization using the BERT model. Subsequently, BERT generates embeddings for the tokenized URL, capturing its contextual and semantic information. These embeddings are then used as features for the classification task. The URL is classified into one of five categories: "Benign", "Defacement", "Phishing", "Malware", or "Spam" based on the learned embeddings and a classification model.

not require the URL to be clicked on; nevertheless, attackers can now easily imitate such features of a benign URL; as a result, this strategy may fail at times. Blacklisting approaches have also been used, but any new malicious URLs that are not already on the list cannot be correctly identified using this technique. None of the reported approaches have focused on addressing the problem of detection of URLs by classifying them into different categories of maliciousness while keeping the semantic meaning of the URLs intact. Therefore, herein, we present a solution for classifying URLs into five different categories with reduced computing cost and minimum feature engineering.

3. Proposed method

This section explains our proposed URL-classification method. The work is presented in the form of a series of subsections that explain various stages of our method. The overall workflow is illustrated in Fig. 2, and the algorithmic details expressed in the form of pseudocode are presented in Algorithm 1. There are three major stages in the proposed URL-classification method: tokenization of URL data, generation of word embeddings, and training/classification. The next subsections provide specific insights into these stages, and an evaluation of the proposed method and a performance analysis are presented in later sections.

3.1. Text preprocessing using BERT tokenizer

The URL data comprise strings of variable length that follow a specific format. These strings are readable and possess semantics, yet they differ from conventional language sentences in many ways; for example, URLs do not contain spaces, and they consist of special characters and punctuation. This difference plays a significant role in the transformation of URL data into a numerical representation, which is a fundamental requirement of the classification task. The first step in the transformation of URL strings into numerical data is to decompose them into semantically independent components, such as words, symbols, and punctuation; the entire process is referred to as tokenization. Conventional string-tokenization approaches, which use white space, commas, or other delimiters to identify word boundaries, are not applicable to URL data as they do not have such word boundaries. We employed the BERT tokenizer to split URLs into constituent units; this tokenizer uses conventional subword embeddings along with novel positional embeddings. Subword embeddings capture the semantics of a word, whereas positional embeddings capture its contextual or order-related information. In combination, these two embeddings can result in better matching of strings or sentences. Specifically, the

Algorithm 1 URL classification using BERT-based model.

Require: Raw URL dataset, Pretrained BERT tokenizer, Label mapping;
Ensure: Trained BERT-based classifier, Evaluation results;

procedure LoadData($URLData$)
  $D \leftarrow URLData$
  $D \leftarrow ClassLabels$
  $D_{train}, D_{val}, D_{test} \leftarrow CreateSplits(D)$
  return $D_{train}, D_{val}, D_{test}$
end procedure
procedure Initialize
  $BERT \leftarrow loadBERT_{pretrained}()$
  $BERT.loadTokernizer()$
  $BERT \leftarrow \eta$ ▷ η is learning rate
  return $BERT$
end procedure
procedure Train($BERT_{classifier}, D_{train}, D_{val}$):
  for each epoch **do**
    $BERT_{classifier} \leftarrow D_{train}$
    $Loss_{train}, Acc_{train} \leftarrow BERT_{classifier}$
    $BERT_{classifier} \leftarrow D_{val}$
    $Loss_{val}, Acc_{val} \leftarrow BERT_{classifier}$
  end for
  return $BERT_{classifier}$
end procedure
procedure Test($BERT_{classifier}, D_{test}$):
  $BERT_{classifier} \leftarrow D_{test}$
  $Loss_{test}, Acc_{test} \leftarrow BERT_{classifier}$
  return $Loss_{test}, Acc_{test}$
end procedure
procedure ClassifyURL($URLData$)
  $D_{train}, D_{val}, D_{test} \leftarrow$ LoadData($URLData$)
  $BERT_{untrained} \leftarrow$ Initialize()
  $BERT_{trained} \leftarrow$ Train($BERT_{untrained}, D_{train}, D_{val}$)
  $Loss_{test}, Acc_{test} \leftarrow$ Test($BERT_{trained}, D_{test}$)
end procedure

BERT tokenizer uses a pretrained transformer model for subword tokenization to divide each word into a series of subwords and then encode it into numerical values. This subword tokenization allows the tokenizer to accurately represent the semantics of a word, even when it is split into multiple tokens.

Figs. 3 and 4 provide an example to emphasize the inefficiency of conventional tokenization and the effectiveness of the BERT tokenizer in handling complex structured URLs. Formally, let $x_i \in X$ represent the ith URL in a collection of URLs X. During tokenization, the BERT tokenizer T decomposes each URL x_i into a sequence of k subwords $t_1, t_2, \ldots, t_k$, each of which is then assigned a distinct numeric value e_j. In this way, the given URL x_i is transformed into a vector representation $v_i = [e_1, e_2, \ldots, e_k]$.

In conventional word-embedding approaches such as those employed by FastText, TF-IDF, word2vec, and GloVe, words lose their contextual meanings. In such approaches, each instance of a word is treated in the same way regardless of its context. Consequently, different semantics of a word are represented by the same vector, and this can cause ambiguous representation of a word's meaning. This problem can be formally stated as follows:

$$\vec{v}\text{bank}_1 = \vec{v}\text{bank}_2, \tag{1}$$

where $\vec{v}\text{bank}_1$ and $\vec{v}\text{bank}_2$ represent the vector representations of the word "bank" in different contexts. BERT addresses this problem by taking into account the words surrounding a target word as well as the wider context of each word's usage; this allows BERT to better capture the context-specific meaning of a word. This can be formally stated as:

$$\vec{v}\text{bank}_1 \neq \vec{v}\text{bank}_2. \tag{2}$$

Thus, BERT-based tokenization can be applied to URLs to prevent the loss of contextual information, and this can significantly improve the overall accuracy of URL classification.

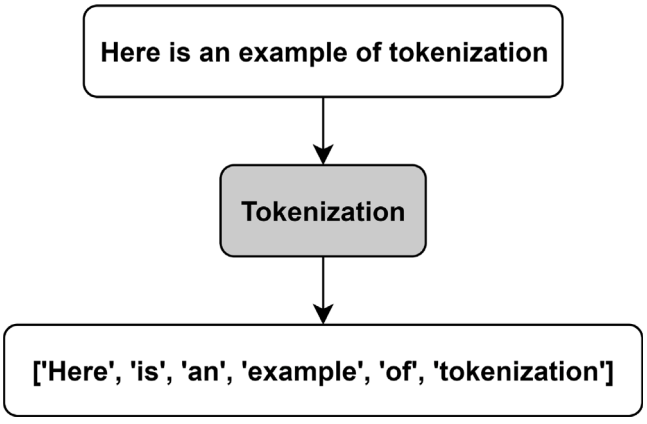

Fig. 3. Word tokenization using white space.

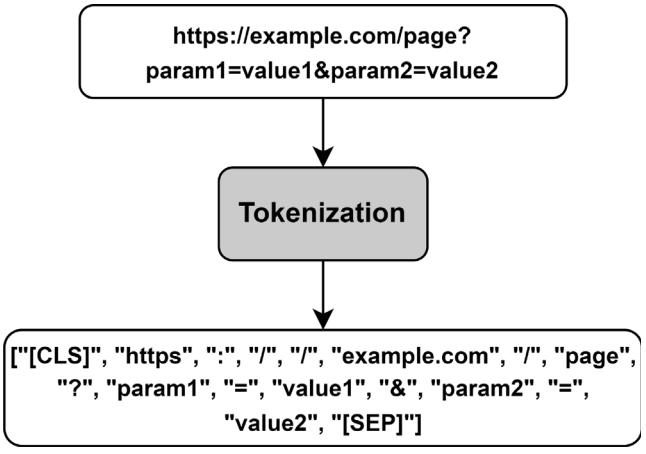

Fig. 4. URL tokenization using BERT format.

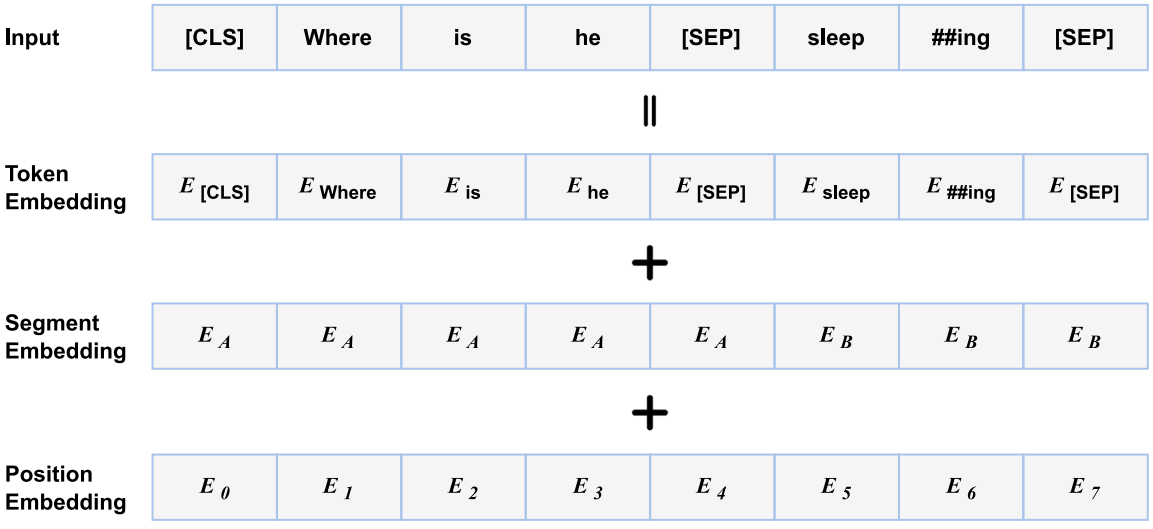

Fig. 5. Input format required by BERT.

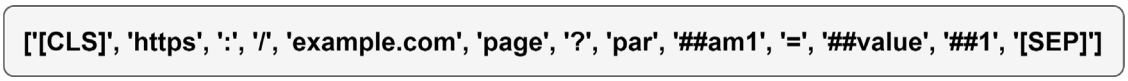

Fig. 6. Embeddings generated through BERT.

3.2. Generating word embeddings

The pretrained deep-learning architecture known as the BERT model needs the input data to be prepared in a particular format to perform well on natural-language-processing (NLP) tasks. As such, before beginning any downstream analysis or modeling, it is essential to transform the input data into the necessary BERT format. Fig. 5 depicts the input format required for BERT, which is made up of three essential elements: token embeddings, segment embeddings, and positional embeddings.

As illustrated in Fig. 5, the language model of BERT uses certain tokens such as $[SEP]$ and $[CLS]$ in its operation. In this case, the $[CLS]$ token is used at the start of a text, and the $[SEP]$ token is used to denote the beginnings or ends of sentences. Regardless of the task for which it is being used, BERT needs both tokens. Using tokens that are compatible with the fixed vocabulary V used by BERT is the next step. Let t_i represent the ith token in an n-length sequence. Each token t_i receives a special token ID, indicated by $w_i \in \mathbb{Z}^{|V_c|}$, where $|V_c|$ is the vocabulary size. To differentiate between tokens and padding elements, BERT uses mask IDs (m_i); similarly, segment identifiers (s_i) are used to distinguish between various sentences. Lastly, the position of each token inside a sequence is indicated by positional embeddings, which are denoted by p_i. For instance, BERT would produce embeddings for the URL "https://example.compage?param1=value1" as shown in Fig. 6.

The special tokens and a vector of sequence tokens make up the two main components of the input layer. BERT employs the WordPiece method for tokenization, which successfully divides tokens such as "playing" into "play" and "##ing". This makes it possible to cover a wider range of out-of-vocabulary words. Let $\mathbb{Z}^{|V|}$ signify the space of token IDs and let T represent the set of tokens in a sequence of length n. Each token $t_i \in T$ has a token embedding $\mathbf{e}_i \in \mathbb{R}^d$ that corresponds to its vocabulary ID w_i. To differentiate between two different sentences, say A and B, a sentence embedding $\mathbf{e}_s \in \mathbb{R}^d$ is used.

Finally, positional embeddings, represented by $\mathbf{e}_p \in \mathbb{R}^d$, are used to determine the position of each word inside a given sequence. Positional embeddings are needed because the processing of tokens is conducted in parallel rather than sequentially, as is the case with LSTM methods. The position of each token is maintained by the position-embedding matrix $\mathbf{P} \in \mathbb{R}^{n\times d}$, and the resultant final input representation of a sequence of tokens is as follows:

$$\mathbf{E} = [e_1, e_2, \ldots, e_n] + [e_p] \tag{3}$$

where $[\cdot]$ denotes concatenation and $\mathbf{E} \in \mathbb{R}^{n\times d}$ is the final input embedding matrix.

3.3. URL classification with BERT

The chosen model, BERT, can have different sizes, each with different numbers of parameters and layers, and different resource requirements. In this context, BERT-Base and BERT-Large are the main categories. As illustrated in Fig. 7, the key distinction between these models lies in their depth, with BERT-Large boasting twice the number of transformer blocks as BERT-Base.[1] BERT-Large comprises 24 transformer blocks, 16 attention heads, and 340 million parameters, while BERT-Base is composed of 12 transformer blocks, 12 attention heads, and 110 million parameters. We found that the BERT-Base model was more efficient than BERT-Large for our URL-classification problem.

The BERT-Base model employs feed-forward neural networks and a self-attention mechanism to process the input sequence. Let $E = [e_1, e_2, \ldots, e_n]$ be the corresponding embeddings obtained from the tokenization procedure, and let $X = [x_1, x_2, \ldots, x_n]$ be the input sequence of length n. The sequence is then transmitted up the stack of L transformers, where each layer carries out the following operations:

$$Z^l = \text{LayerNorm}\left(X^{l-1} + \text{SelfAttention}(X^{l-1})\right), \tag{4}$$

$$X^l = \text{LayerNorm}\left(Z^l + \text{FeedForward}(Z^l)\right), \tag{5}$$

where the input and output of the lth layer, respectively, are X^{l-1} and X^l. The queries $Q = X^{l-1}W_Q$, keys $K = X^{l-1}W_K$, and values $V = X^{l-1}W_V$ are used in the self-attention operation, where W_Q, W_K, and W_V are learnable matrices. After this, residual connection and layer normalization are used to combine the output of the self-attention operation with the input. A linear transformation is applied by the feed-forward neural network, and this is followed by a nonlinear activation function such as the rectified linear unit (ReLU).

The classification model's neural network and BERT layers are initialized once the input sequence has been processed. A dropout layer $D_{p_{\text{drop}}}$ is used with dropout rate p_{drop}; dropout regularization is a technique used in neural-network architectures to decrease the potential for overfitting. In this approach, to add noise and keep the network from depending too much on any one feature

[1] https://arxiv.org/abs/1810.04805

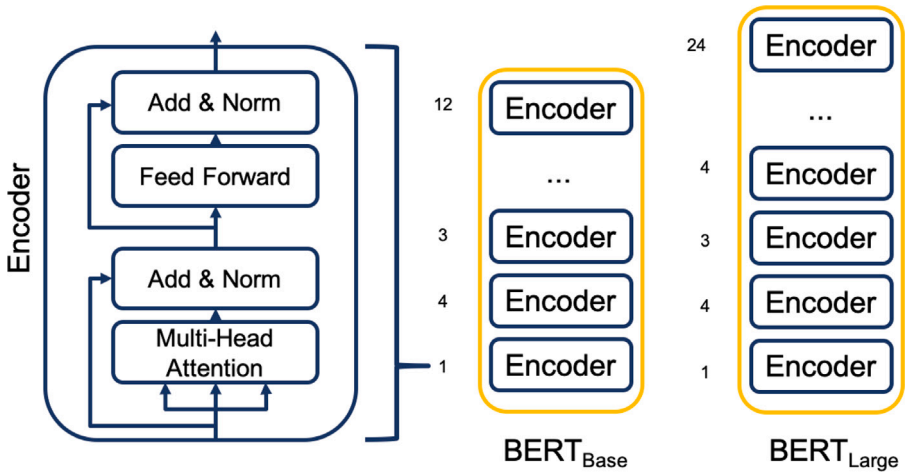

Fig. 7. Architecture of BERT-Base and BERT-Large [27].

during training, a proportion of the neurons are randomly deactivated. To balance the regularization strength and model capacity, the dropout rate, represented as p_{drop}, is carefully selected through empirical experimentation and hyperparameter adjustment. The network's capacity to generalize to previously unseen input is improved via dropout, and this pushes the network to learn more resilient and generalizable representations.

The dense layer, denoted as $D_{5,\text{softmax}}$, comprises five neurons and generates a probability distribution across the five classes by using the softmax activation function. The classification model is then trained by minimizing the sparse categorical cross-entropy loss function, denoted as $L_{\text{sparse\_categorical\_crossentropy}}(Y, \hat{Y})$, where $Y = [y_1, y_2, \ldots, y_c]$ represents the target label distribution, in which c is the number of classes and $\hat{Y}$ is the predicted label distribution. The Adam optimizer and accuracy metric are used during training. To obtain the required outcomes, the model is trained for five epochs because it converges on the fifth epoch. Generally, for a model like BERT, three to five epochs are recommended, and the number is gradually increased to avoid overfitting.

To guarantee neural-network convergence and reliable results, the URL-classification method tracks the convergence and applies early stopping to avoid overfitting. Systematic experimentation is used to evaluate hyperparameters such as batch size and dropout rate. In this way, stability, model capacity, and computing efficiency are balanced while choosing optimal parameters, and these are validated empirically and determined by domain experts. The performance of the BERT-Base model in NLP tasks, along with factors such as processing efficiency and dataset size, inform architectural decisions such as the layer and node counts.

3.3.1. Network design overview

As noted, our URL-classification system employs a transformer-based design using the BERT model. In this system, a 12-transformer-block BERT-Base model selects tokens according to contextual relationships and extracts a large amount of semantic information. It is made up of various layers of feed-forward neural networks and self-attention processes. Feed-forward neural networks process the output of the self-attention layers to improve nonlinearity and identify patterns in the data. The final probability distribution is created using a classification head with a softmax activation function.

The number of layers, the number of nodes per layer, and the number of activation functions of a neural network define its expressiveness and capacity. Each of the 12 layers that make up the BERT-Base model architecture has several feed-forward sublayers and self-attention heads. The dimensionality of hidden representations – whose hidden size is 768 – determines the number of nodes in each layer. To introduce nonlinearity and facilitate data learning, the transformer design makes use of both linear transformations and nonlinear activations (such as ReLU) within the feed-forward sublayers.

The success of the BERT-Base model in NLP tasks, its computational efficiency, and the size of the dataset all have impacts on the architecture of the neural network and the selection of parameters. Each transformer block has 12 layers, balancing model efficiency and capacity, and each layer's 768 neurons correspond to the hidden size of the model. The learning of nonlinear correlations in the data and gradient propagation are facilitated by ReLU activation functions in the feed-forward sublayers. The neural network's efficiency and scalability for URL-categorization tasks are influenced by these design choices.

4. Performance evaluation and results

In this section, we discuss the experiments that we performed to obtain our results. We worked with four different types of embeddings: word2vec, FastText, GloVe, and BERT. Since the URLs are in the form of text, we generated their embeddings to turn them into a numeric form to feed them into our classification model. For classification purposes, we used the BERT layer.

Table 2
Specifications of the computing environment.

Item	Description
Operating system	Windows 10 Pro
Processor	AMD Ryzen Threadripper 3960X 24-Core
RAM	96.0 GB
GPU	NVIDIA GeForce RTX 3060
CUDA version	11.6
Python version	3.9.12

4.1. Experimental work

We performed the experiments on different sizes of dataset to confirm that our proposed approach outperforms all other techniques in each case. In this section, we also explain the evaluation measures that we used to analyze our results. We shall first discuss the experiments, and we shall then proceed to analyze their results.

We used several different metrics for the evaluation of our model, including the accuracy, precision, recall, F-measure, and confusion matrix. We also demonstrated the accuracy of our model on each epoch, as obtained through a graph. All the experiments were performed on a computer running the Windows 10 Pro operating system, and Table 2 presents the computing environment in detail.

4.1.1. Data gathering

As described in detail above, to combat cybersecurity threats posed by rogue websites or URLs, which can result in financial loss, identity theft, and malware installation, we sought to develop a deep-learning-based model that can identify and block harmful URLs. To this end, we compiled two different datasets, which were themselves combined from various sources. The first dataset[2] was taken from Kaggle, which has four categories of URL: benign, defacement, phishing, and malware. Since we were also seeking to work with a fifth class, spam, we also combined this with another dataset.[3] By combining both datasets, we obtained a very large final dataset containing all five classes and comprising 816,557 samples; however, the classes in this combined dataset are very imbalanced, and we thus needed to balance the dataset.

4.1.2. Preprocessing

In the preprocessing of the dataset, we first checked for NaN values, and we then checked the data for any duplicate records. In this exploration, we found no NaN values or duplicate records. To address the imbalance in the dataset, an undersampling technique was applied. The minimum number of records for the spam class is 12,000. We randomly removed records from the remaining four classes – benign, phishing, malware, and defacement – keeping 40,000 samples for each class; however, as we were only able to gather 12,000 spam URLs, and we kept all of these. The final dataset comprises 172,000 records.

We converted our labels into numeric form by mapping them to numbers from 0–4 in such a way that 0 represents benign, 1 represents defacement, 2 represents phishing, 3 represents malware, and 4 represents spam. After this, we split our data into training, testing, and validation sets. The number of records in each dataset is listed in Table 3.

4.1.3. Evaluation of the proposed method

The BERT model needs a token sequence (words) as input. Two special tokens are expected in each sequence: [CLS], the first token that is used for classification, and [SEP], which indicates which tokens belong to which sequence (if there is only one sequence, this is added at the end).

When using the BERT model, we need to tokenize the input text into a sequence of tokens that can be fed into the model. The BERT tokenizer is used to perform this tokenization. Since the BERT model has a fixed input length of 512 tokens, any input text longer than 512 tokens needs to be truncated. Conversely, if the input text is shorter than 512 tokens, we add padding tokens ([PAD]) to the end of the sequence to make its length up to 512. This ensures that all inputs have the same length and can be processed efficiently by the BERT model.

The contextualized information of the input token sequence is captured in the embedding vectors produced by the BERT model, with the [CLS] token's vector representing the entire sequence for text classification. The resulting classifier's output vector has a size of 5, and this represents the probabilities of the input sequence belonging to each of the classes.

The BERT tokenizer is used to conduct all of the necessary text transformations so that the results can be used as input for our BERT model. This automatically adds [CLS], [SEP], and [PAD] tokens. The BERT tokenizer has the following parameters:

- Padding: we use a maximum length of 512 to pad each sequence.
- Max_length: for our dataset, we use 512 because this is the maximum length allowed for working with BERT for a sequence.
- Truncation: we set this parameter as "True" so that the tokens exceeding the maximum length can be truncated.

[2] https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset
[3] https://www.unb.ca/cic/datasets/url-2016.html

Table 3
URL dataset after splitting into training, testing, and validation sets.

Dataset	Benign	Defacement	Phishing	Malware	Spam	Total
Training	31,978	32,129	31,946	31,954	9593	137,600
Testing	4054	3889	3965	4085	1207	17,200
Validation	3968	3982	4089	3961	1200	17,200
Total	40,000	40,000	40,000	40,000	12,000	172,000

- Return_tensors: we set this parameter as "pt" because we are using PyTorch.

The BERT-Base model, which has 12 transformer encoder layers, is used to develop the final model. The output of the model includes two variables: the first stores the embedding vectors of all tokens in a sequence, while the second is the pooled output containing the embedding vector of the [CLS] token. We use this [CLS] token embedding as the input to the classifier, which is sufficient for the task.

After obtaining the pooled output from the BERT model, it is passed through a linear layer with a ReLU activation function. The output of this layer is a vector of size 5, in which each element corresponds to a label category (benign, defacement, phishing, malware, or spam).

The training loop is implemented using PyTorch and follows a typical approach. The model is trained for five epochs using the Adam optimizer with a learning rate of 10^{-6}. As noted earlier, this is trained for five epochs because the model converges on the fifth epoch. Because we are working with multiclass classification, categorical cross-entropy is used as the loss function.

In summary, the input text data are tokenized using the BERT tokenizer, and the tokenized input is then converted into tensors and returned as the input to the ANN. BERT is then used as a feature extractor, and the BERT model is loaded from the `bert-base-cased` pretrained model. The input text data are passed through BERT using the forward method, which retrieves the pooled output from BERT. This pooled output is then passed through a series of fully connected layers (`self.linear` and `self.relu`) to obtain the final layer, representing the predicted class probabilities. The ANN is trained using the `train` function. Within this function, the input data are fed into the ANN. The output of the ANN is compared to the ground-truth labels using the `CrossEntropyLoss` criterion. The gradients are computed, and the optimizer updates the model's parameters to minimize the loss. The `evaluate` function is used to examine the performance of the trained model on the test data. This performs inference using the trained ANN on the test data, computes the predicted labels, and compares them with the ground-truth labels. The function then calculates various evaluation metrics such as the accuracy, precision, recall, and $F1$-score for each class.

4.1.4. Conventional embedding techniques

Three other commonly used embedding techniques in the field of text classification – word2vec, FastText, and GloVe – were also examined so that the results of our approach could be compared with those from these conventional techniques to demonstrate that BERT outperforms them in terms of all evaluation metrics. We also compared our results with those from previous reports [28–30] using different approaches for multiclass classification. Further discussion of the achieved results is presented in the next section.

4.2. Obtained results

In this section, we demonstrate the results of the proposed approach and compare them with other conventional techniques, as well as with the existing literature. We perform experiments using a total of four techniques. Fig. 8 illustrates the training and validation accuracy for each embedding technique.

The first three embedding techniques gave us good results with accuracy values of 0.9239, 0.9485, and 0.9541 for word2vec, FastText, and GloVe, respectively; however, our proposed BERT-based technique outperformed all these techniques and gave the best results, with an accuracy of 0.98. A graphical representation of the results is presented in Figs. 9 and 10 presents the confusion matrix for all the classes obtained when we used the BERT embeddings. From these figures, it can be seen that BERT gives the best results; it obtains very low FPRs and false-negative rates (FNRs), indicating that the obtained results are reliable.

The BERT-embedding technique extracts contextual information and semantic relationships from text better than word2vec, FastText, and GloVe; its transformer architecture, fine-tuning ability, pretrained language model, and bidirectional contextual embeddings all contribute to this exceptional performance. By analyzing a complete "sentence" in both directions, BERT creates contextual embeddings that capture rich contextual information and word dependencies. It can understand complex language patterns and relationships, and this is useful for applications such as URL classification. The transformer architecture of BERT enables long-range dependencies to be captured and sequential data to be processed efficiently. Using its self-attention mechanism, it can generate more informative embeddings by adaptively weighing the significance of various tokens and focusing on relevant parts of an input sequence. By employing supervised learning to refine BERT embeddings on particular tasks, a model can adjust its representations to fit a target domain. In this case, the low FPR and FNR values of the confusion matrix demonstrate how reliable BERT embeddings are and how well they classify URLs into various categories.

Table 4 presents the comprehensive results of all the embeddings, and Table 5 lists the results for individual classes, showing the accuracy, precision, recall, and F-measure for the benign, defacement, phishing, malware, and spam categories. Finally, Table 6 presents a comparison of our results with the those from previous studies in terms of common evaluation metrics.

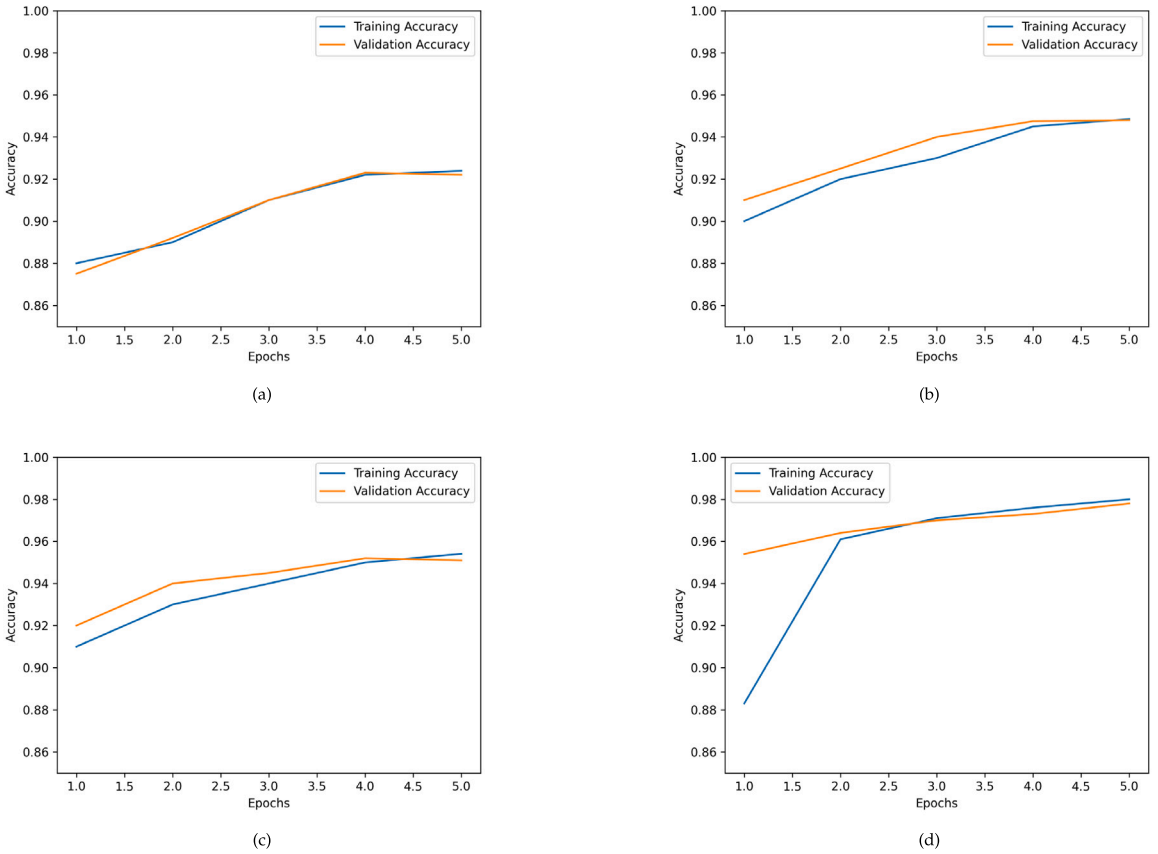

Fig. 8. Training and testing accuracies of various word-embedding methods: (a) word2vec; (b) FastText; (c) GloVe; and (d) BERT. Each plot shows the performance during both the training and testing phases. These visual representations provide valuable insights into the comparative effectiveness of these word-embedding methods for the purpose of malicious-URL classification.

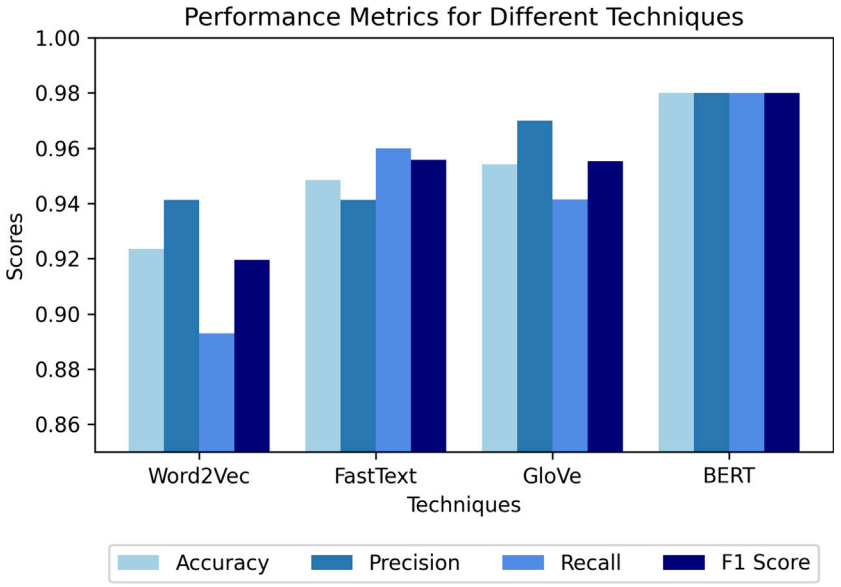

Fig. 9. Results of all embedding techniques.

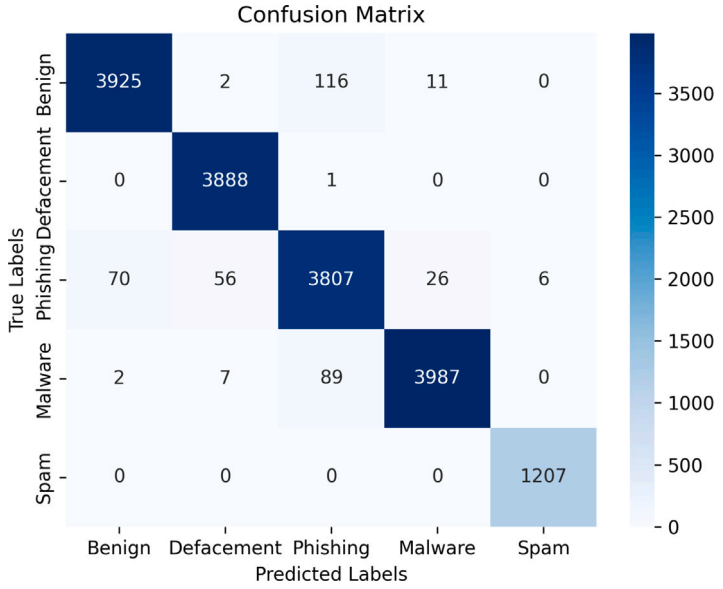

Fig. 10. Confusion matrix obtained using BERT.

Table 4
Results of all embedding techniques.

Embedding	Accuracy	Precision	Recall	F-measure
word2vec	0.9239	0.9412	0.8929	0.9195
FastText	0.9485	0.9412	0.9600	0.9558
GloVe	0.9541	0.9700	0.9414	0.9553
BERT	**0.9804**	**0.9804**	**0.9804**	**0.9804**

Table 5
Results of individual classes in the dataset.

Class	Accuracy	Precision	Recall	F-measure
Benign	0.9664	0.9819	0.9681	0.9750
Defacement	0.9997	0.9835	0.9997	0.9915
Phishing	0.9601	0.9486	0.9601	0.9543
Malware	0.9762	0.9908	0.9760	0.9833
Spam	1.0000	0.9950	1.0000	0.9975

Table 6
Comparison with previous reports (micro average).

Approach	Accuracy	Precision	Recall	F-measure
[28]	0.9792	0.9777	0.9787	0.9782
[29]	0.9698	0.9681	0.9681	0.9781
[30]	0.9537	0.9537	0.9537	0.9537
Proposed	**0.9804**	**0.9804**	**0.9804**	**0.9804**

5. Conclusions and future work

Online social networks have become an integral part of modern life facilitating communication and information sharing on a massive scale. Their vast popularity has also made them prime targets for cybercriminals who use deceptive URLs to exploit users, emphasizing the need for heightened vigilance against such threats. Based on BERT, this paper proposes a technique for identifying fraudulent URLs via a multiclass URL-classification process. Previously reported techniques have relied on static characteristics derived from embeddings such as word2vec, FastText, and GloVe. Such embedding techniques cannot capture the sequential meaning of a URL, require time-consuming manual feature engineering, and are incapable of handling unknown features in URLs containing test data. The model in the present work created embeddings using BERT so that the transformer considers the context of each word based on its adjacent words; therefore, positional embeddings play a key role in the BERT input. The model developed in this work achieved an accuracy of 0.98, with an F-measure of 0.98, a recall of 0.98, and a precision of 0.98. After comparison of our findings

with the results from existing embeddings, it was found that the proposed approach outperformed all of them, including word2vec, FastText, and GloVe.

The model developed in this work could be further enhanced by incorporating the most recent BERT variations, such as roBERTa, which outperforms BERT by a factor of 2% to 20%. A major reason for this performance improvement is that the memory with which it was trained is 144 GB larger than that used for training BERT. Using the roBERTa model could thus boost the performance of the proposed method.

CRediT authorship contribution statement

Sara Afzal: Conceptualization, Data curation, Investigation, Methodology, Visualization, Writing – original draft, Writing – review & editing. **Muhammad Asim:** Conceptualization, Methodology, Supervision, Validation, Writing – review & editing. **Mirza Omer Beg:** Conceptualization, Methodology, Validation, Writing – review & editing. **Thar Baker:** Conceptualization, Methodology, Writing – review & editing. **Ali Ismail Awad:** Conceptualization, Methodology, Funding acquisition, Writing – review & editing. **Nouman Shamim:** Formal analysis, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data used in this study were taken from public datasets.

Acknowledgments

The authors would like to express their sincere gratitude to the anonymous reviewers for their valuable feedback and suggestions, which have improved the quality of this work. This work was in part supported by a joint research grant between United Arab Emirates University and Zayed University (UAEU-ZU) under the Big Data Analytics Center at UAEU, United Arab Emirates, Grant No. 12R141.

References

[1] Huang L, Jia S, Balcetis E, Zhu Q. ADVERT: an adaptive and data-driven attention enhancement mechanism for phishing prevention. IEEE Trans Inf Forensics Secur 2022;17:2585–97.
[2] Liu D, Lee J-H. CNN based malicious website detection by invalidating multiple web spams. IEEE Access 2020;8:97258–66.
[3] Afzal S, Asim M, Javed AR, Beg MO, Baker T. URLdeepDetect: A deep learning approach for detecting malicious URLs using semantic vector models. J Netw Syst Manage 2021;29(3):1–27.
[4] Arin E, Kutlu M. Deep learning based social bot detection on twitter. IEEE Trans Inf Forensics Secur 2023;18:1763–72.
[5] Patil D, Patil J. Feature-based malicious URL and attack type detection using multi-class classification. ISC Int J Inf Secur 2018;10(2):141–62.
[6] Khan F, Ahamed J, Kadry S, Ramasamy LK. Detecting malicious URLs using binary classification through ada boost algorithm. Int J Electr Comput Eng (2088-8708) 2020;10(1).
[7] Thantharate P, Anurag T. CYBRIA-Pioneering federated learning for privacy-aware cybersecurity with brilliance. In: 2023 IEEE 20th international conference on smart communities: improving quality of life using AI, robotics and IoT. IEEE; 2023, p. 56–61.
[8] Roy SS, Awad AI, Amare LA, Erkihun MT, Anas M. Multimodel phishing URL detection using LSTM, bidirectional LSTM, and GRU models. Future Internet 2022;14(11):340.
[9] Bahnsen AC, Bohorquez EC, Villegas S, Vargas J, González FA. Classifying phishing URLs using recurrent neural networks. In: 2017 APWG symposium on electronic crime research (eCrime). IEEE; 2017, p. 1–8.
[10] Johnson C, Khadka B, Basnet RB, Doleck T. Towards detecting and classifying malicious URLs using deep learning. J Wirel Mob Netw Ubiquitous Comput Dependable Appl 2020;11(4):31–48.
[11] Mamun MSI, Rathore MA, Lashkari AH, Stakhanova N, Ghorbani AA. Detecting malicious URLs using lexical analysis. In: International conference on network and system security. Springer; 2016, p. 467–82.
[12] Kumi S, Lim C, Lee S-G. Malicious URL detection based on associative classification. Entropy 2021;23(2):182.
[13] Telo J. Supervised machine learning for detecting malicious URLs: an evaluation of different models. Sage Sci Rev Appl Mach Learn 2022;5(2):30–46.
[14] Chiramdasu R, Srivastava G, Bhattacharya S, Reddy PK, Gadekallu TR. Malicious URL detection using logistic regression. In: 2021 IEEE international conference on omni-layer intelligent systems. IEEE; 2021, p. 1–6.
[15] Rupa C, Srivastava G, Bhattacharya S, Reddy P, Gadekallu TR. A machine learning driven threat intelligence system for malicious URL detection. In: The 16th international conference on availability, reliability and security. 2021, p. 1–7.
[16] Wejinya G, Bhatia S. Machine learning for malicious URL detection. In: ICT systems and sustainability. Springer; 2021, p. 463–72.
[17] Aljabri M, Alhaidari F, Mohammad R, Mirza S, Alhamed D, Altamimi H, Chrouf SM. An assessment of lexical, network, and content-based features for detecting malicious URLs using machine learning and deep learning models. Comput Intell Neurosci 2022;2022.
[18] Chawla A. Phishing website analysis and detection using machine learning. Int J Intell Syst Appl Eng 2022;10(1):10–6.
[19] He S, Li B, Peng H, Xin J, Zhang E. An effective cost-sensitive XGBoost method for malicious URLs detection in imbalanced dataset. IEEE Access 2021;9:93089–96.
[20] Le H, Pham Q, Sahoo D, Hoi SC. URLNet: learning a URL representation with deep learning for malicious URL detection. 2018, arXiv preprint arXiv:1802.03162.
[21] Rao RS, Umarekar A, Pais AR. Application of word embedding and machine learning in detecting phishing websites. Telecommun Syst 2022;79(1):33–45.

[22] Do NQ, Selamat A, Krejcar O, Yokoi T, Fujita H. Phishing webpage classification via deep learning-based algorithms: an empirical study. Appl Sci 2021;11(19):9210.
[23] Saxe J, Berlin K. eXpose: a character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. 2017, arXiv preprint arXiv:1702.08568.
[24] Prabakaran MK, Meenakshi Sundaram P, Chandrasekar AD. An enhanced deep learning-based phishing detection mechanism to effectively identify malicious URLs using variational autoencoders. IET Inf Secur 2023;17(3):423–40.
[25] Patgiri R, Biswas A, Nayak S. deepBF: malicious URL detection using learned bloom filter and evolutionary deep learning. Comput Commun 2023;200:30–41.
[26] Bo W, Fang ZB, Wei LX, Cheng ZF, Hua ZX. Malicious URLs detection based on a novel optimization algorithm. IEICE Trans Inf Syst 2021;104(4):513–6.
[27] Kumar MB. BERT variants and their differences. 2023, URL https://360digitmg.com/blog/bert-variants-and-their-differences.
[28] Abu Al-Haija Q, Al-Fayoumi M. An intelligent identification and classification system for malicious uniform resource locators (URLs). Neural Comput Appl 2023;1–17.
[29] Manyumwa T, Chapita PF, Wu H, Ji S. Towards fighting cybercrime: Malicious URL attack type detection using multiclass classification. In: 2020 IEEE international conference on big data (big data). IEEE; 2020, p. 1813–22.
[30] Uçar E, Ucar M, İncetaş MO. A deep learning approach for detection of malicious URLs. In: 6th international management information systems conference. 2019, p. 10–7.

Sara Afzal (M.Sc.) is a Lecturer in the Department of Cyber Security at the National University of Computer and Emerging Sciences, Islamabad, Pakistan. She received her M.Sc. in Computer Science from the same university. Her research interests include Cyber Security, Blockchain, and Machine Learning.

Muhammad Asim (Ph.D.) is a Professor in Cyber Security at the National University of Computer and Emerging Sciences, Pakistan. He earned his Ph.D. from Liverpool John Moores University and an M.Sc. in Computing from Liverpool University. With over fifteen years of experience, he specializes in Cyber Security, IoT, Cloud Computing, Applied AI/ML, and Blockchain research and development.

Mirza Omer Beg (Ph.D.) is an Artificial Intelligence professional with extensive experience in the field. He currently holds a position as an NLP/NLU expert at Afiniti. Dr. Beg has a wealth of expertise in the field of NLP/NLU and spearheads the development of advanced conversational AI solutions.r.

Thar Baker (Ph.D.) (Senior Member, IEEE) (Senior Fellow, HEA) is a Professor of Industry 4.0/5.0 at the University of Brighton, UK. He earned his Ph.D. in Autonomic Cloud Applications from Liverpool John Moores University, where he was later promoted to reader in cloud engineering and became head of the applied computing research group. His research interests lie in the areas of cloud computing, edge AI, IoT, sensor networks, and federated learning.

Ali Ismail Awad (Ph.D.) is currently an Associate Professor of Cybersecurity at the College of Information Technology, United Arab Emirates University (UAEU), Al Ain, United Arab Emirates. He is also an Honorary Associate Professor at the University of Nottingham, UK. His research interests include cybersecurity, network security, Internet of Things security, and image analysis with biometrics and medical imaging applications.

Nouman Shamim (Ph.D.) is an Assistant Professor at the Department of Computer & Information Sciences, PIEAS, Islamabad. He received his Ph.D. degree in Computer Science from the National University of Computer and Emerging Sciences, Pakistan. His research interests include Cyber Security, Blockchain, and machine learning.