# Robust Detection of Malicious URLs With Self-Paced Wide & Deep Learning

Yunji Liang, Qiushi Wang, Kang Xiong, Xiaolong Zheng, Zhiwen Yu, *Senior Member, IEEE*, and Daniel Zeng, *Fellow, IEEE*

**Abstract**—As cybercrimes grow in scale with devastating economic costs, it is important to protect potential victims against diverse attacks. In spite of the diversity of cybercrimes, it is the uniform resource locators (URLs) that connect vulnerable users with potential attacks. Although numerous solutions (e.g., rule-based solutions and machine learning-based methods) are proposed for malicious URL detection, they cannot provide robust performance due to the diversity of cybercrimes and cannot cope with the explosive growth of malicious URLs with the evolution of obfuscation strategies. In this paper, we propose a deep learning-based system, dubbed as *CyberLen*, to detect malicious URLs robustly and effectively. Specifically, we use factorization machine (FM) to learn the latent interaction among lexical features. For the deep structural features, position embedding is introduced for token vectorization to reduce the ambiguity of URL tokens. Meanwhile, temporal convolution network (TCN) is utilized to learn the long-distance dependency among URL tokens. To fuse heterogeneous features, self-paced wide & deep learning strategy is proposed to train a robust model effectively. The proposed solution is evaluated on a large-scale URL dataset. Our experimental results show that position embedding is constructive to reducing the ambiguity of URL tokens, and the self-paced wide & deep learning strategy shows superior performance in terms of F1 score and convergence speed.

**Index Terms**—Self-paced learning, temporal convolutional network, factorization machine, wide & deep, malicious URL detection

## 1 Introduction

With the popularity of social media platforms, cybercrimes grow in scale with huge economic costs and large-scale data breaches for devastating security risks[1], [2], [3]. It is projected that global cybercrime costs will reach up to $6 trillion annually with a per-minute global cost of $11.4 million by 2021[1]. In spite of the diversity of cybercrimes, it is the uniform resource locators (URLs) that serve as the "gateway" to bridge the compromised users with the malicious agents (e.g., ransomware, maleware, and trojan) [2], [4], [5]. As shown in Fig. 1, vicious URLs that direct to the malicious agents are disguised with comprised websites and forged accounts [6]. Before clicking them, it is difficult for the potential victims to tell whether a URL is malicious or benign. Thus, detecting malicious URLs in advance is an indispensable task to protect the vulnerable users against cyberattacks.

To detect malicious URLs, URL blacklists are widely adopted in the main-stream web browsers to prevent users from visiting vicious websites. The blacklist-based solutions need to maintain a huge blacklist, which is looked up to determine whether a URL is benign or not. The blacklist-based solutions are easy to implement, but they are ineffective when faced with the explosive growth of user-generated content for two reasons. On one hand, the explosive growth of malicious URLs makes it difficult to maintain a up-to-date blacklist. For example, it is reported that phishing URLs grow rapidly, with a 400% increase in URLs discovered from January to July in 2019[2]. Meanwhile, the short life span of malicious URLs makes it infeasible to timely update the backlists [7], [8]. On the other hand, it takes great efforts to build a large-scale blacklist with intensive labor and huge time consumption. For example, there are several crowdsourcing websites (e.g., *https://www.phishtank.com*) to flag malicious URLs discovered by crowds. However, the flagged URLs eventually have to be evaluated by moderators. The training costs of moderators are huge for organizations [9].

To detect malicious URLs automatically, numerous studies rely on feature engineering to construct explanatory variables with knowledge of data, and utilize classical machine learning algorithms to recognize malicious URLs. With regard to feature engineering, lexical features (e.g., the number of special characters, the length of a URL, $n$-gram

1. https://www.helpnetsecurity.com/2020/08/28/global-cost-of-cybercrime-per-minute/

2. https://www.helpnetsecurity.com/2019/10/09/phishing-increase-2019/

- *Yunji Liang, Qiushi Wang, Kang Xiong, and Zhiwen Yu are with the School of Computer Science, Northwestern Polytechnical University, Xi'an, Shaanxi 710060, China. E-mail: {liangyunji, zhiwenyu}@nwpu.edu.cn, wqiushi@mail.nwpu.edu.cn, 3412638248@qq.com.*
- *Xiaolong Zheng and Daniel Zeng are with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation Chinese Academy of Sciences, Beijing 100080, China. E-mail: {xiaolong.zheng, dajun.zeng}@ia.ac.cn.*

*Manuscript received 12 Jan. 2021; revised 12 Oct. 2021; accepted 13 Oct. 2021. Date of publication 20 Oct. 2021; date of current version 14 Mar. 2022.*
*This work was supported in part by the National Key Research and Development Program under Grant 2018AAA0100500, the Natural Science Foundation of China under Grant 61902320, and Fundamental Research Funds for the Central Universities under Grant 31020180QD140.*
*(Corresponding author: Yunji Liang.)*
*Digital Object Identifier no. 10.1109/TDSC.2021.3121388*

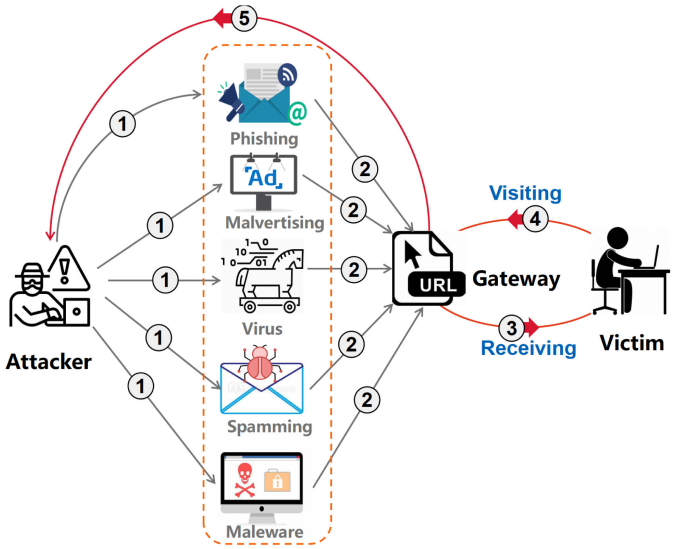


Fig. 1. The process of attacks from the adversary. In spite of the diversity of cybercrimes, URLs are the 'gateway' to bridge the adversary and the victims.

features) are extracted to characterize URLs. However, lexical features are helpless with the development of obfuscation strategies. Obfuscation strategies are widely used by attackers to evade static analysis on lexical URL features by disguising malicious URLs as the benign ones statistically [7], [9], [10]. To enrich the feature representation of URLs, embedding strategies on word level and character level are applied to capture both the lexical features and the structural features [11]. With regard to the classification strategies, numerous classification algorithms including support vector machine (SVM), decision tree, random forest and Naïve Bayes are adopted for the detection of malicious URLs [12], [13]. However, machine learning-based solutions heavily rely on the feature engineering. As attackers also continuously adjust their tactics over time, invalidating some of the learned information [9], [14] and integrating emerging information, the models trained based on the short life-span training dataset do not work well for the emerging malicious URLs.

Recently, deep neural networks (DNN) including convolutional neural networks (CNN) and recurrent neural network (RNN) are gaining popularity, and achieve the state-of-the-art performances in numerous classification tasks. For example, Pham *et al.* fused CNN and long short-term memory (LSTM) with character-level word embedding for URL classification [11]. George *et al.* applied the generative adversarial network (GAN) to detect phishing attacks [15]. However, extant DNN-based solutions for URL classification tasks typically treat URLs as textual data, learn the deep representation of URLs on the level of tokens and apply deep learning models directly without considering the unique patterns of URLs. Prior solutions do not fit well with URL classification tasks for the following reasons.

- *Position Sensitivity*. URL tokens are position sensitive. In general, a URL can be divided into two sections: *domain* section and *path* section. The same token located in different sections has different meanings. For example, in $http://paypal.org/paypal/gateway.html$, *paypal* in domain section is an identifier for a specific website, which indicates where one given URL is hosted. While *paypal* in the path section refers to the relative path. Numerous studies showed that well-known tokens are placed in different sections to misguide vulnerable users [7], [16]. In addition, statistically, over 75% of malware on Windows systems hides in one of three subdirectories: %temp%, %appdata% and %cache% [3]. Obviously, URL tokens are position sensitive and the position information is crucial for malicious URL detection. However, prior studies treat URLs as textual data and overlook the importance of token position for URL classification.
- *Long-distance Dependency*. Around 25% of malicious URLs are hosted on trusted domains, as the trusted domains are less likely to raise suspicion and are more difficult to block[4]. To address this problem, the dependency of tokens will be helpful for malicious detection. Although RNN and its variants including LSTM and gated recurrent unit (GRU) are widely used for sequential inputs, the training costs for RNN-based models are huge with energy draining concerns and long time consumption [17]. An efficient neural network to learn long-distance dependency with lower training consumption will be promising for timely malicious URL detection.
- *Heterogeneous Features*. With regard to the URL representation, we can characterize a URL from multiple perspectives with heterogeneous features including lexical features and structural features. How to build one robust model based on heterogeneous features for malicious detection is challenging.

To address the aforementioned problems, we investigate the position embedding of URLs, efficient DNN models for sequential data, and the fusion strategy of heterogeneous features for malicious URL detection. The main contributions of this paper are shown below.

- Given that tokens with different positions have different functional features, we introduce the position embedding of URL representation to reduce the token ambiguity. Unlike prior studies that treat URLs as textual data, the position embedding of URL representation taps into the structural patterns of URLs and distinguishes the tokens according to their structural functions. To the best of our knowledge, this is the first work to investigate the position embedding of URLs.
- Due to the disadvantages of CNNs and RNNs, we propose an interpretable model to learn the long-distance dependency with lower complexity. Specifically, we use temporal neural network (TCN) to learn the structural features. On one hand, we take advantage of the deeper network architecture and parallel computation of TCN to cope with the sequential inputs for less computational complexity.

3. https://www.carbonite.com/blog/article/2019/11/businesses-beware-watch-out-for-these-recent-malware-trends
4. https://www.helpnetsecurity.com/2019/10/09/phishing-increase-2019/

On the other hand, the residual connection is integrated into TCN to learn the context dependency among URL tokens, which improves the interpretability of deep-learning based models.

- To fuse the heterogeneous features, we introduce one self-paced wide & deep learning system, dubbed as *CyberLen*, to incorporate the statistical, lexical and structural features for malicious URL detection. The wide component in *CyberLen* captures the latent interaction among statistical and lexical features. The deep counterpart characterizes the deep structural features. To improve the robustness of model to be trained, the self-paced learning with diversity (SPLD) [18] is applied to ensure the proposed solution works well for both easy and diverse samples.

The remainder of this paper is organized as follows. In Section 2, we summarize the prior studies associated with URL representation and malicious URL detection. After formulating the research problem in Section 3, in Section 4 we highlight the system framework and present the details of position embedding, factorization machine, and temporal convolutional network, respectively. In Sections 5 and 6, we describe our experimental settings and present the primary experimental results. Finally, Section 7 offers concluding remarks to summarize the main contributions in this paper.

## 2 Related Work

In this section, we conduct a systematic overview about the studies for malicious URL detection from the perspectives of URL representation and classification algorithms.

### 2.1 Decomposition of URLs

A URL specifies the location of a web resource and provides a mechanism to retrieve the corresponding Internet information. As shown in Fig. 2, typically, a URL can be decomposed into four parts: *protocol*, *domain name*, *path*, and *file*. The *protocol* part indicates which protocol should be used to access the information specified in the domain. The *domain name* allows users to visit websites by remembering an easy set of words or other characters instead of a long string of numbers. The *domain name* is made of two parts: the top level domain (TLD) and *subdomain*. As shown in Fig. 2, *.com* is the TLD and *progresltda* is the *subdomain*. The *subdomain* along with TLD forms a fully qualified domain name that can be used to visit the website. The *path* is a list of subdirectories from the root of the server to the file. For the *file* part, it usually refers to the name of resources, and sometimes it also includes a list of parameters.

### 2.2 URL Representation

URL representation aims to construct one feature map to characterize a URL instance. The quality of URL representation is significantly important for malicious URL detection. The URL representation can be characterized from different perspectives including lexical and structural patterns.

#### 2.2.1 Lexical Features

The lexical features including the length of a URL, the frequencies of delimiters, special characters (e.g., '-', '@', 'http', and 'www'), and ratio features are widely applied to characterize URLs [14], [19]. For example, Zhu *et al.* [20] found that the average domain/path token length and brand name presence are discriminative features for the detection of malicious URLs. For a phishing URL, it is more likely to contain target brand name to make the vulnerable users to trust the forged link. Meanwhile, numerous studies found that the length of URLs, special symbols (such as @) [16], the number of TLD in a URL are useful for malicious URL detection.


Fig. 2. The decomposition of a URL. In general, a URL is consisted of 4 parts including protocol, domain name, path, file name.

In addition, numerous studies treat URLs as text sequences and apply the language models to characterize a URL. Specifically, numerous studies model URL tokens with the "bag-of-words" model, and extract lexical features including *n*-gram features and term frequency–inverse document frequency (tf-idf) for URL classification. Experimental results show that *n*-gram features are efficient features for URL classification tasks [21], [22]. However, lexical features only focus on the co-occurrence of tokens and overlook the position information of tokens. For lexical features, the hand-crafted feature engineering is labor intensive and time consuming [23]. Meanwhile, the lexical features extracted from one dataset may introduce bias when they are applied to other datasets, which finally will result in the biased performance of trained models [14].

#### 2.2.2 Structural Features

Several studies established that the ordering, especially the position of tokens in URLs matters. Liang *et al.* analyzed the relative positions of tokens to the TLD, and found that token position was constructive for the detection of malicious URLs [24]. Yan *et al.* studied the domain representation learning and found the vectorized domain features are useful for the detection of malicious URLs [25]. To achieve URL vectorization, a URL is split by delimiters and is fed to a word2vec model to learn the word-level embedding [26]. Similarly, the character-level representation of URLs is applied as well for the detection of malicious URLs and found that deep-learning models along with character-level embedding performed best [11]. URLNet takes advantages of word-level and character-level embedding representations for malicious URL detection [27]. However, word2vec and character-level embedding are based on the co-occurrence of words/tokens and overlook the importance of position information of tokens. The overlook of position information may lead to the ambiguity of tokens, and cannot discriminate the subtle differences of tokens. How to embed the position information in URL vectorization is still an open problem.

### 2.3 Methods for Malicious URL Detection

Numerous methods have been proposed for the detection of malicious URLs. Generally, extant solutions can be grouped

into three categories: blacklist-based methods, machine learning-based methods and deep learning-based methods.

#### 2.3.1 Blacklist-Based Methods

For the blacklist-based methods, they maintain one blacklist with a large number of known malicious URLs. In the mainstream web browsers, they provide blacklist extensions to block the requests when a vulnerable user is going to visit a suspicious website. Although the blacklist-based solutions are effective to find the suspicious URLs by matching the blacklist, they are fragile when faced with the explosive growth of URLs for the following reasons [28]. First, updating blacklists is very slow, resulting in that blacklist-based solutions do not work well for the emerging suspicious URLs [29]. Second, it is found that malicious URLs have very short time span to evade the detection [7], [8], [30]. The nature of malicious URLs makes it extremely infeasible to maintain an up-to-date blacklist. Third, blacklist-based solutions are static with no learning capability and have lower robustness when faced with the URLs generated by new evasion techniques [31], [32].

#### 2.3.2 Machine Learning-Based Methods

Unlike blacklist-based solutions, a large number of machine learning-based models have been applied to learn the generalized models based on the existing suspicious URLs. Many classification algorithms including random forest, support vector machine, decision tree, kNN, maximum entropy [33] are applied. Ma *et al.* [34] used statistical methods to find the attributes of malicious URLs, extracted nine features, and compared the proposed classification model with Naïve Bayes, SVM, and logistic regression. The dominant factor determining the model performance is feature engineering [35]. Prior studies rely on hand-crafted features, which is time consuming and tricky to build one complete feature space. The static feature engineering does not work well with the explosive growth of URLs.

Machine learning-based methods aim to learn a static model from the whole observation samples. However, they overlook the complexity and diversity of samples, and take all the samples equally for model training. To address these problems, online learning has become a promising technique to learn the emerging patterns from continuous streams of data [36]. Ma *et al.* fused both the lexical and host-based features of URLs and proposed a confidence-weighted online learning classifier to detect malicious URLs [16], [37].

#### 2.3.3 Deep Learning-Based Methods

Recently, deep neural networks are gaining popularity for URL classification tasks. Sameen *et al.* proposed PhishHaven [26] to detect malicious URLs generated by a deep neural network-based phishing URL generating system called DeepPhish. PhishHaven employed lexical analysis for feature extraction and used the ensemble-based machine learning models for AI-generated phishing URLs detection [26]. Numerous studies used convolutional neural network (CNN) to learn the deep representation of URLs and applied the RNN or LSTM for the classification of sequential input [11], [29], [38]. For example, Yang *et al.* proposed a malicious keyword based representation of URLs to learn features both on character and token level, used CNN to learn the deep representation of URLs, and applied the gated-recurrent-unit (GRU) neural network for the detection of malicious URLs [29]. Similarly, Rajalakshmi *et al.* used a recurrent convolutional neural network (RCNN) to recognize the adult websites only from URLs [38]. To overcome the limitations of deep-learning models on training dataset, Trevisan *et al.* [39] used generative adversarial neural networks (GAN) for URL classification. They claim that GANs perform well when faced with a limited training set and even for not appeared URLs [39]. However deep learning-based models are subject to adversary attacks. AlEroud *et al.* used GANs to generate phishing URLs to bypass the AI-based phishing detection models, and found that it is effective to create phishing URL to evade the detection [15]. Meanwhile, heterogeneous features are applied to improve the accuracy of URL classification. Yang *et al.* proposed a multidimensional feature phishing detection approach for URL classification. They first extracted character-level features from URLs and fused extra features (e.g., URL statistical features, webpage code features, webpage text features) for URL classification [35].

However, the aforementioned studies did not take the uniqueness of URLs into account. In this paper, we study the explainable models for the detection of malicious URLs. Specifically, we study the representation learning of URL on the token level to embed the position information, and investigate the deep learning-based models to improve the robustness of models on diverse samples. To achieve this goal, we propose a self-paced wide & deep framework to fuse both lexical features and structural features by considering both the complexity and the diversity of samples during model training.

## 3 Problem Formulation

Unlike prior studies that mainly focus on the lexical features, in this paper we investigate how to fuse heterogeneous features including statistical features, lexical features and deep structural features and aim to provide an interpretable model for the detection of malicious URLs. Our research problem in this paper can be formulated as follows. Given a URL dataset $\mathcal{E} = \{(U_i, y_i)\}_{i=1}^{M}$, we first extract the lexical feature vector $\mathbf{x}_i = \{x_1, \ldots, x_p\}$ for each URL instance $U_i$, where $\mathbf{x}_i$ is a $p$-dimensional feature vector and $p$ is the number of lexical features; $y_i$ is the label information of sample $U_i$. Next we aim to find one mapping function $\hat{y}_i = f_1(\mathbf{x}_i)$ with the maximum likelihood estimation of $y_i$; For the deep structure features, we split each URL instance $U_i$ with the delimiters. After the token separation, the original URL $U_i$ can be denoted by a sequence of tokens $U_i = [u_1 :, \ldots, : u_n]$, where $u_i$ refers to one token and $n$ indicates the number of tokens in the trimmed URL. The sequence of tokens are fed to the *embedding* layer to learn the vector representation of each token. After token vectorization, a trimmed URL can be denoted as $\mathbf{U}_i = [\mathbf{u}_1; \mathbf{u}_2; \ldots, \mathbf{u}_n]^{\mathsf{T}}$, where $\mathbf{u}_i$ is the vectorization representation of $i$th token. Next, we aim to find one objective function $f_2(\mathbf{U}_i)$ close to the ground truth $y_i$. After the extraction of statistical features and deep structure features, the co-training strategy is introduced to fit both

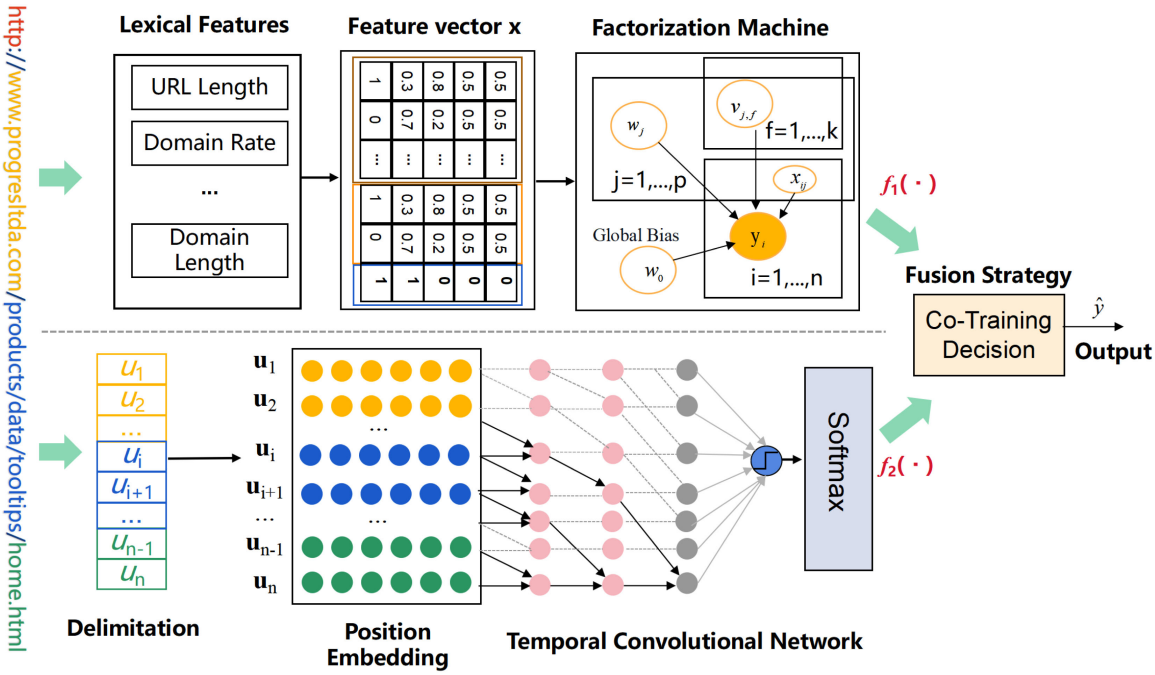


Fig. 3. Overview of self-paced wide & deep framework. The top branch refers to the wide component to characterize the lexical features and statistical features, and utilizes the factorization machine to learn the latent interaction among features; While the bottom branch adopts the position embedding and temporal convolution network to learn the deep structural features of URLs. The self-paced learning strategy is introduced to co-train the two branches effectively.

lexical features and deep structural features. To achieve this goal, we apply the self-paced learning with diversity [40] for the co-training of two branches to improve the robustness and the efficiency.

## 4 Methodology

In this section, we propose a deep learning-based system, dubbed as *CyberLen*, to discriminate malicious URLs robustly and effectively. The critical point of *CyberLen* is the self-paced wide & deep framework that fuses heterogeneous features and considers both the sample complexity and sample diversity. Fig. 3 illustrates the overall framework of *CyberLen*.

### 4.1 System Framework

To detect the emerging malicious URLs, we propose *CyberLen*, a generic framework to fuse heterogeneous features including lexical features, statistical features and deep structural features. As shown in Fig. 3, the proposed framework follows the wide & deep architecture [41], where two branches are introduced to capture both the statistical features and structural features respectively. The top branch in Fig. 3 (the wide component) applies the feature engineering to capture lexical features and statistical features of URLs, and uses factorization machine [42] to learn latent interaction among features. The bottom branch in Fig. 3 (the deep component) takes advantages of the position embedding to learn the fine-grained representation of URLs by introducing the position information of tokens into representation learning. To learn the deep structural features, the deep component utilizes a temporal convolution network [43] to uncover the structural features in URLs.

For the wide component, feature engineering is conducted for each URL instance, where numerous features including lexical features and statistical features are extracted. Next, we can build the feature vector $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i$ is the $p-$dimensional feature vector of $i$th URL. As the extracted features are not strictly independent and the sparsity of feature distribution, we use factorization machine to characterize the lexical features with the following advantages including efficient training consumption and mining the latent interaction among features [42].

For the deep component, we extract the tokens of URLs by a set of delimiters. After the delimitation, the original URL can be treated as a sequence of tokens. Based on the observation that brands are more likely to be inserted in the phishing URLs to deceive the potential victims [7], [20], we introduce the position embedding to reduce the ambiguity of tokens by taking the positional information of tokens into consideration. To capture the deep structural features of tokens, we propose the temporal convolution network to learn the long-distance dependency of tokens.

Fusion strategy aims to fuse the heterogeneous features extracted from the conventional encoder and the deep encoder. Due to the diversity and the evolution of malicious URLs, the fusion component should be able to cope with the diversity of URLs and changes of malicious URLs. To achieve these goals, we adopt the self-paced learning with diversity (SPLD) [44] to deal with the diverse samples and subtle changes.

### 4.2 Position Embedding

It is observed that well-known tokens (such as *www*, *http*) and brand names (e.g., *chase*, *paypal*) are more likely to be

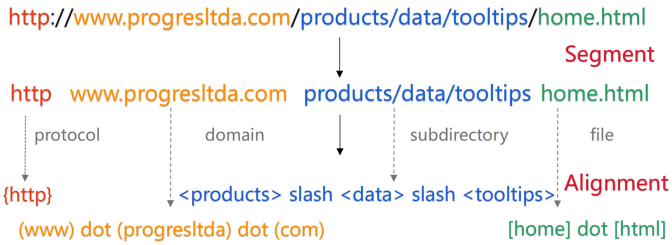


Fig. 4. An example of token extraction. A URL is decomposed into four chunks. Alignment strategies are conducted for each chuck to minimize the token ambiguity.

presented in *directory* part of malicious URLs to deceive the vulnerable users [7], [20]. In addition, the token semantic changes according to changes of token positions. For example, the token *chase* in *chase.com.chasebankingonline.usa.cc/login.html* refers to the domain name. While the token *Chase* in *211.95.3.83/Chase/page/* indicates the subdirectory. Due to the importance of token position for malicious URL detection, we propose the position embedding of URLs to minimize the ambiguity of tokens.

The workflow of position embedding is shown in Fig. 4. For a large-scale URL dataset, we first conduct data preprocessing according to the following rules.

- The sequential strings following # are removed, as # in a URL refers to one section of webpage and will not be send to the server end.
- The sequential strings following ? are removed, as ? in a URL indicates that the web browser should not use the cached information.

After the data pre-processing, we perform token extraction to obtain tokens from URLs. First, we slice a URL into four chunks: *protocol*, *domain*, *path*, and *file* by the position of /. The chunk before the first / is taken as *protocol* part; The strings before the second / are defined as *domain part*; the strings following the last / are considered as *file part*. The remainder strings are treated as *path part*. Next, the trimmed sequential strings are split by a set of delimiters, which are defined in $\mathcal{A} = \{/, \backslash, -, :, ., \#, @, ?, =\}$. To minimize the information loss, we introduce special tokens including *slash*, *equal*, *dot*, *hyphen*, *hash*, *dash*, *at*, *mark*, *colon* to replace the special delimiters defined in $\mathcal{A}$. To overcome the token ambiguity problem, the alignment strategy locates tokens in different chucks with different types of brackets. The processing is illustrated in Fig. 4. Specifically, each token in the *protocol* part is placed in curly brackets {}; while each token in the *domain* part is placed in parentheses (), and tokens in the *path part* are placed in angle brackets $< >$. Tokens in *file* part are decorated with square brackets [].

After token extraction, the generated token sequences are fed to a word2vec model [45]. Word2vec is a two-layer neural network and is widely applied in natural language processing to embed words in a lower-dimensional vector space, where vectors close together in vector space have similar meanings based on context and word-vectors distant to each other have differing meanings. In this paper, we follow the idea of word2vec to learn the vector representation of URL tokens to capture the syntactic and semantic relationships among URL tokens. In practice, we use the skip-gram model [45] to train the vectorized representation of URL tokens.

### 4.3 Wide Component With Factorization Machines

For the feature engineering, we choose a set of lexical features that are widely used in prior studies [7], [16] to characterize the statistical features of a given URL. The brief descriptions of these lexical features are shown below.

- *Features related with tokens:* Tokens are extremely important by providing the semantic features. Given a URL, the number of token and the average length of tokens in each subsection are used to characterize a URL. In addition, as malicious URLs are apt to contain extremely long tokens, the length of the longest token is quantified to find the abnormal tokens as well.
- *Features related to length:* Significant differences are observed in terms of URL length. Malicious URLs are getting longer due to addition of variables or malicious redirection [7]. Length of each subsection in a given URL is measured.
- *Features related with digits:* Malicious URLs are more likely to insert digits in URLs to make it difficult for victims to understand. We use the digit count to measure the frequency of digits in different subsections respectively. In addition, number rate calculates the proportion of digits in the different subsections of URLs.
- *Features related with letters:* Following the features related with digits, we calculate the statistical features of letters in a URL similarly. Letter count measures how many letters are used in a specified URL.

As the aforementioned lexical features may not be independent and extremely sparse, we apply factorization machine (FM) [42] to learn the high-order latent interaction among lexical features. A 2-order FM model can be formulated as Eq. (1), where $w_0$ is the model bias; $w_i \in \mathbb{R}$ models the weight of variable $x_i$; $\langle \mathbf{v}_i, \mathbf{v}_j \rangle = \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f}$ characterizes the pairwise interaction between variables $x_i$ and $x_j$. The pairwise interactions can be reformulated as Eq. (2) [42], [46]. Compared with SVM, FM shows great advantages. First, SVM assumes that all features are independent and independently contribute to the probability of the target variable's class [42]. While FM does not need this independent assumption on features and can be used to model the interactions among features. Second, FM models all nested variable interactions with a factorized parametrization instead of a dense parametrization like in SVMs. This ensures that FM has linear time complexity in $O(kn)$.

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \tag{1}$$

$$\begin{aligned} &\sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \\ &= \frac{1}{2} \sum_{f=1}^{k} \left( \left( \sum_{i=1}^{n} v_{i,f} x_i \right)^2 - \sum_{i=1}^{n} v_{i,f}^2 x_i^2 \right) \end{aligned} \tag{2}$$

For ML based classification algorithms, the conditional independence assumption states that features are conditional independent of each other given the conditioning

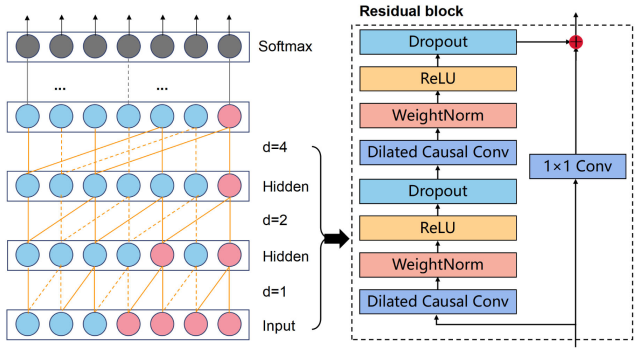


Fig. 5. Basic framework of temporal neural network. The residual block is introduced in TCN to build deeper neural network.

variable. The independence assumption among features is very strong, and is hardly ever true for URL lexical features. To this end, we use FM to weaken the tight assumption of feature independence by digging out pairwise interactions between features. For a classification task, the FM model is trained by minimizing the cross-entropy loss as shown in Eq. (3), where $p_i$ is the predicted probability distribution and $\hat{y}_i$ refers to the predicted result.

$$loss = \sum_i y_i log p_i, where p_i = \frac{1}{1+e^{\hat{y}_i}} \tag{3}$$

### 4.4 Deep Component for Feature Representation

With the popularity of deep learning in numerous fields, different types of neural networks such as convolutional neural networks (CNN) and recurrent neural networks (RNN) have achieved great success. However, CNN cannot encode time-series information and has small receptive field; RNN cannot preserve long-distance dependency with huge training consumption in terms of time and space complexity. Lately, temporal convolutional network (TCN) is proposed to address the aforementioned problems. As shown in Fig. 5, the generic framework of TCN is consisted of three components: *causal convolution*, *dilated convolution*, and *residual connection*.

The causal convolution depends only on the values of the input at the present time and in the past. Formally, a causal convolution operation is one mapping function $f(x_{(1)}, \ldots, x_{(t)}) \rightarrow y_{(t)}$ that satisfies the causal constraint that $y_{(t)}$ depends only on $x_{(1)}, \ldots, x_{(t)}$ and not on any "future" inputs $x_{(t+1)}, \ldots, x_{t+n}$ [47]. In order to build deeper networks, residual blocks as shown in Fig. 5 are introduced in TCN to increase the network depth for the improvement of network expression ability [48]. Numerous studies applied the residual connection to build deeper network with significant performance gain. The skip connections in the residual blocks facilitate preserving the norm of the gradient and lead to stable back-propagation [49].

Compared with RNN and its variants, TCN not only shows competitive performances in terms of accuracy and model size, it presents great advantages in several aspects including parallelism, stable gradients, and low memory requirements [50], [51]. First, TCN fixes the vanishing & exploding gradient problems and does not require time back-propagation during the training process. Second, dilated convolution in TCN makes the receptive field grow exponentially with the depth of the network, which makes it possible to build deeper neural networks. Third, TCN has a higher degree of parallelism than RNN, which makes their training and deployment more computationally feasible. In addition, TCN is able to learn the long-distance dependency from sequential input by using a hierarchy of temporal convolutional filters, pooling, and upsampling. Thus, we take the advantages of TCN to learn the structural features of URLs.

### 4.5 Fusion Strategies

To fuse the lexical features and structural features, we introduce three fusion strategies to weight the importance of different features for malicious URL detection.

Given the training dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in \mathbb{R}^m$ denotes the $i$th sample with label information $y_i$. The loss between ground truth $y_i$ and the estimated label $\hat{y}_i^w$ caused by wide component is denoted with $\mathcal{L}_w^i(y_i, \hat{y}_i^w)$. Similarly, the loss of deep component is denoted with $\mathcal{L}_d^i(y_i, \hat{y}_i^d)$, where $\hat{y}_i^d$ refers to the predicted result of the deep component for the $i$th sample.

#### 4.5.1 Classical Fusion

For the simplicity of fusion strategy, we introduce one classical fusion strategy by combing the losses of wide component and deep component with even weights. The classical fusion strategy is formulated as Eq. (4).

$$loss = \sum_{i=1}^{n} (\mathcal{L}_w^i(y_i, \hat{y}_i^w) + \mathcal{L}_d^i(y_i, \hat{y}_i^d)) \tag{4}$$

#### 4.5.2 Self-Paced Leaning

We apply the self-paced learning to fuse the heterogeneous features. Self-paced learning is an emerging learning method inspired by the learning process of human and animals that gradually incorporates easy to more complex samples into training [44]. Given the training dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^n$ represents the features for the $i$th URL in $\mathcal{D}$, and $y_i$ are the corresponding class for the $i$th URL. To fuse both the lexical features and the structural features, self-paced learning strategy aims to co-train the parameters $\mathbf{w}$ of the wide & deep model and learn the latent weight variable $\mathbf{v} = [v_1, \ldots, v_n]$ by minimizing Eq. (5).

$$\begin{aligned} \min_{\mathbf{w},\mathbf{v}} \mathbb{E}(\mathbf{w}, \mathbf{v}; \lambda) = &\sum_{i=1}^{n} v_i (\mathcal{L}_w^i(y_i, \hat{y}_i^w) + \mathcal{L}_d^i(y_i, \hat{y}_i^d)) \\ &- \lambda \sum_{i=1}^{n} v_i \end{aligned} \tag{5}$$

In Eq. (5), parameter $\lambda$ controls the learning rate. $\mathcal{L}_w$ refers to the losses from the wide part quantified by logit loss; $\mathcal{L}_d$ denotes as the losses from deep part measured by cross-entropy loss. The objective of SPL is to minimize the weighted training loss together with the negative $l_1$-norm regularizer [44]. To solve Eq. (5), an iterative biconvex optimization strategy is applied [40], [52]. Its primary idea can be described as follows: When $\mathbf{v}$ is fixed, Eq. (5) can be

treated as one standard supervised learning objective function to learn the parameters $\mathbf{w}$ with the minimal losses. Next, for a given $\mathbf{w}$, the global optimum $\mathbf{v} = [v_1^*, \ldots, v_n^*]$ can be calculated by Eq. (6), where $v_i^*$ is a gate unite to control whether one sample should be selected in the training dataset according to its easiness.

$$v_i^* = \begin{cases} 1, & \mathcal{L}_w^i(y_i, \hat{y}_i^w) + \mathcal{L}_d^i(y_i, \hat{y}_i^d) < \lambda \\ 0, & otherwise \end{cases} \tag{6}$$

#### 4.5.3 Self-Paced Learning With Diversity

Although SPL is able to cope with both the simply and complex samples, it does not consider the diversity of URL samples. Prior studies show that the trained models could be biased with significant performance differences using different samples. To this end, self-paced learning with diversity (SPLD) [40] is proposed and can be formulated by Eq. (7).

$$\begin{aligned} \min_{\mathbf{w},\mathbf{v}} \mathbb{E}(\mathbf{w}, \mathbf{v}; \lambda, \gamma) = & \sum_{i=1}^{n} v_i(\mathcal{L}_w^i(y_i, \hat{y}_i^w) + \mathcal{L}_d^i(y_i, \hat{y}_i^d)) \\ & - \lambda \sum_{i=1}^{n} v_i - \gamma \sum_{j=1}^{b} \left\| \mathbf{v}^j \right\|_2 \end{aligned} \tag{7}$$

Different from Eq. (5), one more regularization term is introduced in Eq. (7), where $\gamma \sum_{j=1}^{b} \|\mathbf{v}^j\|_2$ refers to the diversity term and $b$ is the number of partitions during the training. To address this problem, Meng *et al.* proposed one efficient algorithm for obtaining the global minimum. For more details, please refer to study in [40]. The SPLD strategy is able to control how many easy samples are selected, and also balance the diversity of samples among different partitions.

## 5 Experiments

### 5.1 Dataset

To evaluate the proposed solution, we synthesize a large-scale dataset by integrating samples from multiple data sources. Specifically, we build crawlers to collect the reported suspicious URLs from pishtank[5] and openphish[6] respectively. Meanwhile, we gather together several public datasets including whitelist[7], Shalla's blacklist[8], phishing army[9], and two more public datasets downloaded from github[10,11].

We combine all the data sources into one large-scale dataset. For the synthetic dataset, we conduct data cleaning including removal of duplicated samples, data trimming and data formatting. For data trimming, we remove the strings following ?, as the question mark is used as a separator in a URL to denote the start of query string. Next, data formatting is conducted to label the trimmed samples. Specifically, the dataset is formatted into two columns. The trimmed URLs are placed in the first column. Labels of URLs are organized in the second column. As the categorization information is inconsistent, we build the mapping strategy among datasets. We encode the samples according to the obfuscation strategies enumerated in pishstorm [30]. In our experiment, we mainly focus on five obfuscation strategies including *URL obfuscation with other domain*, *URL obfuscation with keywords*, *typosquatting domains or long domains*, *URL obfuscation with IP address* and *obfuscation with URL shortener*.

After data cleaning, finally we get 1,299,110 URL instances. Among them, 819,110 URLs are labeled as malicious. We shuffle the dataset and randomly select 960,000 instances as training dataset. To recognize malicious URLs, we formulate the problem in this paper as a 6-class classification problem. To evaluate the performance of the proposed solution, we turn to accuracy, precision, recall, and $F1$ score to quantify the performances.

### 5.2 Comparison Models

To evaluate the performance of our proposed solution, we choose the widely used machine learning classification algorithms including kNN, decision tree, random forest, logistic regression, SVM, adaboost and gradient boosting as baselines. In addition, as URLs are treated as text inputs in most prior studies, we also choose classical text classification algorithms as baselines. The brief descriptions of baselines are shown below. For the fairness, we use the features defined in Section 4.3 to train machine learning-based models. For deep learning-based models, we adopt the position embedding to achieve token vectorization.

- *TextCNN* was first proposed for text classification in [53], where the weights of words are assigned according to the pre-trained vectors, and can be dynamically tuned during training.
- *TextRNN*: RNN and its variants including LSTM and GRU are widely applied to learn the dependency in sequential data, and show great advantages in numerous tasks. In this paper, we choose the bidirectional LSTM (Bi-LSTM) as the baseline.
- *RCNN* [54] applies a bi-directional recurrent structure to capture contextual information as far as possible when learning word representations, and utilizes a max-pooling layer to capture the key clues in text.
- *FastText* [55] utilizes the bag-of-words of $n$-gram features instead of word vectors to capture partial information about the local word order. Then the averaged text representation is fed to a hierarchical softmax to speed up the training process.
- *DPCNN* [56] integrates the region embedding and shortcut connection in low-complexity word-level CNN for text classification to efficiently represent long-range associations in text.
- *Transformer* [57] utilizes self-attention of words, positional embedding and multi-head attention to long-term dependency. Transformer is independent from

5. http://phishtank.org/
6. https://openphish.com/
7. https://github.com/anudeepND/whitelist
8. http://www.shallalist.de/
9. https://phishing.army/
10. https://github.com/savan77/Malicious-URL-Detection-using-Machine-Learning
11. https://github.com/faizann24/Using-machine-learning-to-detect-malicious-URLs

TABLE 1
Tokens With Highest Similarity in Different Sections

| Token | Domain | Path | Difference |
|---|---|---|---|
| *com* | nocovernightclubs, pjkb, dananeer, elitetv, goldvideocenter, ebanqsandbox | mazarbhai, pierce, gatto, willowsoul, tulsaworld, magicstarsystem | 0.008 |
| *web* | jdis, offerdeck, pinedaleonline, roasted, ecologique, zvideo | cinos, miniplay, binarybud, cr2bt, imua, jazztelia | 0.005 |
| *www* | backbonejs, dipac, erstream, epolylearning, ista, nippyfish | kettleriverfurn, kingbreezely, cocorico, programa, mtnl, mouille | 0.002 |
| *login* | stiltskin, maynard, investharborside, thelegacyreport, feedabrain, armut | rcc, mobning, nokiame,interview, sielok, hemawewe | 0.001 |
| *paypal* | webo, maxfocus, ttwschool, otokudays, arihantspot, freesexwebcams | seborrheic, alswalf, webtvhd, iessay, partnership, tutor4quran | 0.005 |
| *chase* | stochastique, adistalkshow, competitiveexam, mmsp, amoshack, iqhealth | friarsonbase, harryshempstore, shugarysweets, magebees, exterasafe, jolla | 0.002 |
| *de* | rendersmart, summerhills, shopoglot, zamanova, ttshow, cartel | gamessakhr, vipbcp, pipetrakit, belkin, purrcenter, truekytrot | 0.002 |

RNN and CNN models, and is highly parallelism with low computational complexity.

### 5.3 Experimental Settings

The following experiments are conducted on a server running the RedHat 6.5 operating system with two Quad Intel Xeon Platinum 8260 @ 2.4 GHz processors, 1024 GB RAM and three NVIDIA TESLA V100 PCIe GPU accelerators. We implement all the baselines and the proposed algorithm in PyTorch. The source code is available at Github[12].

## 6 Experimental Results

### 6.1 Position Embedding of URLs

According to prior studies, malicious URLs are more likely to include the well-known domain tokens in the path level to deceive potential users clicking the links. To emphasize the impacts of token positions for malicious URL detection, we introduce the position embedding of URLs. To evaluate the effects of position embedding for ambiguity reduction, we compare the differences of token vectors w/o position embedding. Specifically, we first select the top 10 most frequent tokens that appear both in the *domain* section and the *path* section. Then, for each token, we retrieve the most similar tokens of each token when it appears in *domain* section and *path* section respectively.

The results are shown in Table 1. Accordingly, we find that when token *com* appears in the *domain* section, the most similar tokens for *com* include *nocovernightclubs*, *elitetv*, *and ebanqsandbox*. When token *com* appears in the *path* section, it is similar to *mazarbhai*, *gatto* and *willowsoul*. For the token *paypal*, it has the highest similarity with *webo*, *maxfocus*, *ttwschool*, and *freesexwebcams* in the *domain* section. While it shows high association with *seborrheic*, *webtvhd*, *iessay*, and *partnership* in *path* section. Obviously, with the change of position for a specific token, the two sets of similar tokens are significantly different. Meanwhile, we compare the token similarity when appearing in different sections of URLs. As shown in Table 1, the representation vectors of token *www* when presenting in different sections have low similarity. There is significant difference between the tokens with $\rho < 0.05$. This indicates that tokens appearing in different sections should be treated as two tokens to avoid the token ambiguity. Based on the aforementioned analysis, we come to the conclusion that URL tokens are position sensitive and should not be treated as text sequences directly.

To quantify the differences of tokens that may present both in *domain* and *path* sections, we define a token set $\mathcal{B} = \{w_1, \ldots, w_i, \ldots w_n\}$, where $w_i$ is a token that appears in the domain level and the path level. For each token $w_i \in \mathcal{B}$, it can be quantified by $\langle \mathbf{v}_i^d, \mathbf{v}_i^p \rangle$, where $\mathbf{v}_i^d$ is the vector representation when the token $w_i$ occurs in the domain section, and $\mathbf{v}_i^p$ is the corresponding vector when $w_i$ appears in the path section. We measure the difference of each token by cosine similarity (See Eq. (8)).

$$sim = \frac{\mathbf{v}_i^d \cdot \mathbf{v}_i^p}{\|\mathbf{v}_i^d\| \|\mathbf{v}_i^p\|} \tag{8}$$

The distribution of token similarities is shown in Fig. 6. It is obvious that the difference of vector representation for one given token is significant. The majority of tokens that present both in *domain* and *path* sections have low similarities, the values of which belong to $[-0.2, 0.2]$. Up to 80% tokens have similarities within $[-0.1, 0.1]$. The representation vectors of corresponding tokens are significantly different when one token appears in different sections of the URL. To minimize the token ambiguity in different contexts, we use position embedding to embed the position information in token vectorization by considering the functions of URL sections.

### 6.2 Ablation Analysis

To study the impacts of position embedding, wide component, deep component and different fusion strategies on the overall performance, we conduct ablation analysis to answer the following questions:

- *Q1:* How does each component impact the overall performance respectively?
- *Q2:* Is position embedding helpful for malicious URL detection?

12. https://github.com/shiqiuwang/wide_deep_with_self_paced_learning

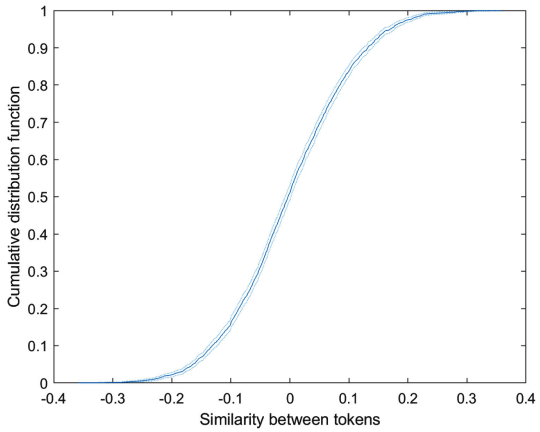


Fig. 6. Cumulative distribution of token similarity.

- *Q3:* Is self-paced learning strategy constructive for the dynamic changes of malicious URLs?

For the simplicity, we examine the performance of wide component (FM), deep component (TCN), and the combinations of wide and deep components with different embedding strategies and different fusion strategies. For *Q1*, we explore the performances by integrating different components. For *Q2*, we compare the effects of word2vec (*W*) and position embedding (*P*) in the deep component substitutively to evaluate the effects of embedding strategies on the overall performance. For the word2vec embedding, we follow the same strategies for data pre-processing as shown in Section 4.2. After the data pre-processing, the token list is fed to gram-skip model to learn the vector representation for each token. For *Q3*, we apply different fusion strategies defined in Section 4.5 to study the performances under different fusion strategies. The results are shown in Table 2.

#### 6.2.1 Component Analysis

For *Q1*, we reveal the effects of lexical features and structural features on model performance. As shown in Table 2, when only lexical features are used for malicious URL classification, FM performs well with up to 88.50% in terms of accuracy. This indicates that lexical features are constructive for URL classification. When only deep structural features are extracted by deep component, TCN outperforms FM with a 6.73% margin in terms of accuracy. This demonstrates that deep structural features are crucial as well for malicious URL classification. When combining both the lexical and deep structural features, FM-TCN significantly overwhelms both FM and TCN. The overwhelming performance of FM-TCN demonstrates that fusing both lexical and structural features will significantly improve the performance.

#### 6.2.2 URL Representation

For *Q2*, we compare the model performances by replacing word2vec with position embedding respectively. According to Table 2, for one given deep model, the model trained with position embedding outperforms its counterpart with word2vec. For example, FM-TCN with position embedding shows a 1.53% performance gain against the one with word2vec in terms of $F1$ score. The similar trends hold true for other models. This demonstrates that position embedding is constructive for the detection of malicious URLs and is helpful to reduce the ambiguity of tokens by considering the functional semantics of URLs.

#### 6.2.3 Fusion Strategy

To cope with the diversity of samples, in this paper we apply self-paced learning strategy to fuse the wide component and the deep component. To demonstrate the efficiency of the proposed fusion strategy, we investigate the performances of different strategies including standard loss, self-paced learning, and self-paced learning with diversity.

According to Table 2, we find that the fusion strategies including SPL and SPLD provide significant performance improvements against the standard fusion method. For example, the accuracy of FM-TCN with position embedding is 98.28%. A 1% performance gain is achieved with the introduction of self-paced fusion strategies. This observation demonstrates that self-paced learning strategies including SPL and SPLD are constructive to combing two heterogeneous models. In addition, SPLD outperforms SPL with subtle performance gain. This is due to the fact that SPLD considers both the sample complexity and sample diversity for training. During the model training, SPLD is

TABLE 2
Ablation Analysis About Wide and Deep Components

| Embedding | Model | Metrics (%) | | | |
|---|---|---|---|---|---|
| | | *Accuracy* | *Precision* | *Recall* | $F1$ |
| - | FM | 88.50 | 85.90 | 88.50 | 85.02 |
| [0pt]Word2Vec | TCN | 95.23 | 90.65 | 84.73 | 87.23 |
| | FM-TCN | 98.21 | 93.37 | 88.78 | 90.81 |
| | FM-TCN-SPL | 98.92 | 95.13 | 92.11 | 93.49 |
| | FM-TCN-SPLD | **98.97** | **95.21** | **92.37** | **93.67** |
| [0pt]Position Embedding | TCN | 95.63 | 94.30 | 88.02 | 90.71 |
| | FM-TCN | 98.28 | 94.80 | 90.41 | 92.34 |
| | FM-TCN-SPL | 99.22 | 97.25 | 94.75 | 95.92 |
| | FM-TCN-SPLD | **99.27** | **97.42** | **95.44** | **96.38** |

TABLE 3
Comparison of the Proposed Solution Against State-of-the-art Benchmark Algorithms

| Model | Metrics (%) | | | |
|---|---|---|---|---|
| | *Accuracy* | *Precision* | *Recall* | $F1$ |
| Naïve Bayes | 80.11 | 56.67 | 62.12 | 51.76 |
| Logistic Regression | 90.71 | 74.63 | 62.12 | 65.51 |
| KNN | 95.08 | 86.22 | 80.64 | 83.09 |
| Decision Tree | 96.02 | 87.97 | 87.25 | 87.60 |
| Random Forest | 96.62 | 93.65 | 87.17 | 90.04 |
| SVM | 94.01 | 89.97 | 71.68 | 77.15 |
| TextCNN | 88.40 | 88.69 | 88.41 | 87.00 |
| RCNN | 84.60 | 81.15 | 84.60 | 82.80 |
| FastText | 80.83 | 79.18 | 80.83 | 78.89 |
| DPCNN | 93.60 | 82.87 | 72.53 | 76.14 |
| TextRNN | 96.77 | 96.17 | 92.92 | 94.45 |
| Transformer | 94.59 | 94.10 | 89.28 | 91.51 |
| **FM-TCN-SPLD** | **99.27** | **97.42** | **95.44** | **96.38** |

TABLE 4
Effects of Parameters $\lambda$ and $\gamma$ on Model Performance

| $\lambda$ | $\gamma$ | Metrics (%) | | | |
|---|---|---|---|---|---|
| | | *Accuracy* | *Precision* | *Recall* | $F1$ |
| 0 | 0.2 | 98.98 | 97.84 | 97.88 | 97.80 |
| 0.05 | 0.2 | 99.01 | 97.85 | 97.82 | 97.77 |
| 0.1 | 0.2 | **99.07** | 97.48 | 97.43 | 97.34 |
| 0.15 | 0.2 | 99.03 | **98.16** | **98.14** | **98.11** |
| 0.2 | 0.2 | 99.04 | 97.85 | 97.81 | 97.77 |
| 0.25 | 0.2 | 98.99 | 97.98 | 97.96 | 97.93 |
| 0.15 | 0 | **99.06** | 97.93 | 97.90 | 97.86 |
| 0.15 | 0.1 | 99.01 | 97.96 | 97.91 | 97.88 |
| 0.15 | 0.2 | 99.03 | **98.16** | **98.14** | **98.11** |
| 0.15 | 0.3 | 99.05 | 97.57 | 97.55 | 97.40 |
| 0.15 | 0.4 | 99.02 | 97.55 | 97.46 | 97.38 |

apt to learn easy samples first. When the complexities of samples are same, SPLD will choose instances with diverse label information for model training and finally will improve the model robustness by overcoming the performance bias on one specific class.

### 6.3 Model Performance

We compare the performance of *CyberLen* against baselines. The experimental results are shown in Table 3. Although numerous studies show that FastText can be applied to text classification tasks with high accuracy and excellent efficiency, our result demonstrates that FastText does not work well for malicious URL detection. This dilemma could be due to the following facts: First, the co-occurrence of tokens in URL is lower, which makes the bag-of-words model of FastText less efficient for URL representation. Second, the token number of a URL follows the heavy-tail distribution and the majority of URLs have less than 8 tokens. Less token number in a URL makes it challenging for FastText to recognize malicious URL based on $n$-gram features.

According to the experimental results, the CNN-based models including TextCNN, RCNN and DPCNN have poor performance. The $F1$ scores of CNN-based models are lower than 87%. The recall scores of CNN-based models are lower than 89%. These observations demonstrate that CNN-based models have poor performance for malicious URL detection tasks. This is due to the fact that the short URL length makes the convolutional operation less efficient to obtain the features by sliding the kernel on the sequential data. In addition, padding is widely adopted in CNN-based models to format the variable-length sequence data into a fixed-length vector. The padding solutions may impair the overall performance of CNN-based models. By contrast, textRNN and transformer show comparative performances. In our experiment, we choose the variant of TextRNN, Bi-LSTM, as the benchmark. The experimental results show that the $F1$ scores of Bi-LSTM and transformer reach up to 94%.

Compared with all the baselines, our proposed solution FM-TCN-SPLD shows superior performance with a 5% performance margin. Our experimental results show that the lexical features and structural features are constructive for the malicious detection. In addition, the fusion of deep component and wide component with SPLD can improve the classification performance by modulating the weights of features.

### 6.4 Parameter Optimization

In SPLD, the parameter $\lambda$ indicates the how many easy samples are selected from the training dateset and the parameter $\gamma$ refers to what extent the diversity of samples are considered in the training. To examine the effects of parameters $\lambda$ and $\gamma$ of SPLD for model performance, we conduct comparative experiments by changing the values of $\lambda$ and $\gamma$. The results are shown in Table 4.

To evaluate the effects of $\lambda$ on model performance, we fix the value of $\gamma$ as 0.2. According to Table 4, we find that with the increase of $\lambda$ ranging from 0 to 0.15, the overall performance of FM-TCN-SPLD is becoming better. On the contrary, the model performance is degrading by increasing $\lambda$ from 0.15 to 0.25. That means when the diversity of samples is fixed, the introduction of easy samples are constructive for the performance improvement at the beginning of model training. However, more easy samples will not improve the model performance. The high easiness will lead to the overfitting problems on the easy samples, and impair the model performance on complex samples.

Next, we fix the value of $\lambda$ as 0.15, and change the value of $\gamma$ ranging from 0 to 0.4. As shown in Table 4, it is obvious that with the increase of $\gamma$ from 0 to 0.2, FM-TCN-SPLD shows superior performance. While with the increase of $\gamma$ from 0.2 to 0.4, the performance is impaired. This observation demonstrates that the diversity of samples is very important for the performance improvement by training a robust model. Based on these experimental results, we find that it is a trade-off between sample complexity and sample diversity to improve the model performance. The parameter $\lambda$ will avoid the overfitting problem on easy samples by considering the complex samples. The parameter $\gamma$ will control the diversity of samples to increase the robustness of the model to be trained.

### 6.5 Convergence Speed

In Section 6.2.2, we find the position embedding is constructive for performance improvement. To provide reasonable

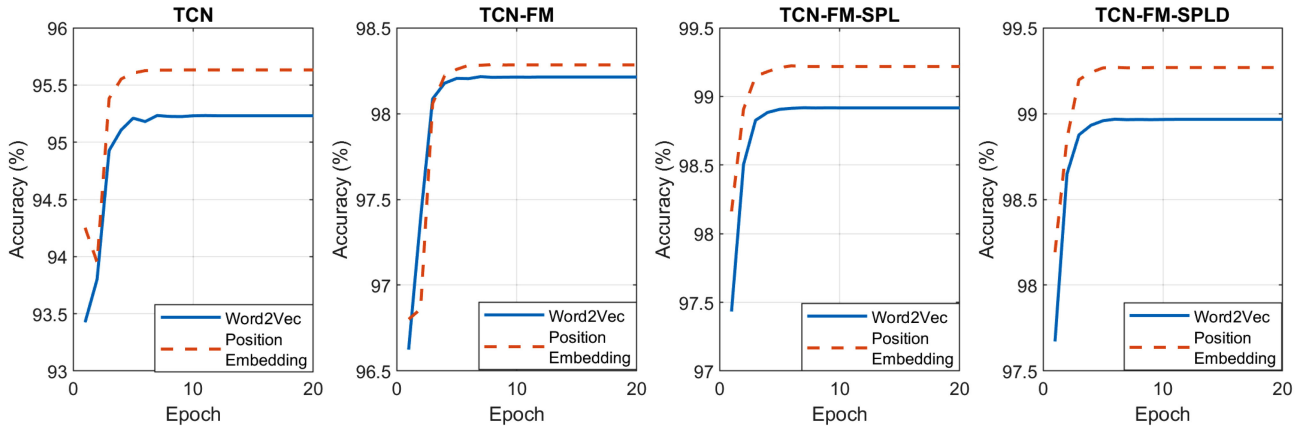


Fig. 7. Comparative studies about the effects of word2vec and position embedding. Different URL representation strategies are used to train models. The changes of model performance with the increase of epoches are shown for four models.

explanation for this conclusion, we conduct comparative studies to explore how the embedding strategy will impact the model performance. Specifically, given one model to be trained, we use word2vec and position embedding to learn the token vectorization respectively. We record the trends of model performance with the increase of epoches.

As shown in Fig. 7, given one specific model, the models with position embedding show superior performance against their counterparts trained with word2vec. These results are consistent with the observation discovered in Section 6.2.2. Furthermore, the models trained with position embedding are apt to converge fast. The fast convergence speed is crucial for model training on a large-scale dataset with lower training costs and less computational load.

### 6.6 Applications

In this paper, we propose a self-paced wide & deep framework to cope with the dynamic and short life-span malicious URLs. Specifically, we use factorization machine to learn the latent interaction among lexical features and employ the temporal neural network to capture the deep structural features with the position embedding. To fuse the heterogeneous features, we apply the self-paced learning strategy to consider both the easiness and the diversity of samples for model training. The detection of malicious URLs is very crucial to ensure the security of cyberspace and is widely applied in many fields including e-business, Internet of things, stocking, etc. Our study can be integrated into existing cybersecurity systems to recognize the suspicious URLs. One the other hand, the proposed self-paced wide & deep framework is a generic method and can be employed to cope with other problems for the following reasons.

- The wide & deep framework was first introduced for recommendation tasks and was demonstrated that it is able to learn the latent interaction among features [41]. In this paper, we are inspired by prior studies, and apply the wide & deep framework for malicious URL detection. The scalability of this framework ensures the model generality by introducing task-specific components.
- To improve the robustness of the model to be trained, we use the self-paced learning strategies to balance the easiness and the diversity of samples. In general, the fusion strategies can be formulated as the regularization terms in the loss functions defined in Eqs. (5) and (7). The regularization terms could be applied into different tasks to balance the easiness and the diversity of samples.

## 7 Conclusion

URLs serve as the gateway to direct the vulnerable users to potential threats. To avoid the traps of cybercrimes, the detection of malicious URLs is urgent. However, the uniqueness of malicious URLs including short-life span, dynamics changes with different obfuscation strategies poses great challenges for extant solutions. To address this problem, we investigate the position representation learning of URLs and propose the position embedding to reduce the ambiguity of tokens, and introduce a self-paced wide & deep framework to cope with the diversity of URLs and enhance the interpretability. Specifically, we utilize the factorization machine to learn the latent interaction among heterogeneous statistical features, and apply the temporal convolutional network for the URL classification task. The experimental results show that the self-paced wide & deep solution has superior performance for the detection of malicious URLs with robust performance and high efficiency.

## References

[1] A. Abbasi, F. M. Zahedi, and S. Kaza, "Detecting fake medical web sites using recursive trust labeling," *ACM Trans. Inf. Syst.*, vol. 30, no. 4, pp. 1–36, Nov. 2012.
[2] T. Li, G. Kou, and Y. Peng, "Improving malicious URLS detection via feature engineering: Linear and nonlinear space transformation methods," *Inf. Syst.*, vol. 91, 2020, Art. no. 101494.
[3] A. Abbasi, Z. Zhang, D. Zimbra, H. Chen, and J. F. Nunamaker, "Detecting fake websites: The contribution of statistical learning theory," *MIS Q.*, vol. 34, no. 3, pp. 435–461, Sep. 2010.
[4] C.-T. Huang, M. N. Sakib, C. A. Kamhoua, K. A. Kwiat, and L. Njilla, "A Bayesian game theoretic approach for inspecting web-based malvertising," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 6, pp. 1257–1268, Nov./Dec. 2020.
[5] G. Sun and Q. Qian, "Deep learning and visualization for identifying malware families," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 1, pp. 283–295, Jan./Feb. 2021.

[6] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna, "Towards detecting compromised accounts on social networks," *IEEE Trans. Dependable Secure Comput.*, vol. 14, no. 4, pp. 447–460, Jul./Aug. 2017.
[7] M. S. I. Mamun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, and A. A. Ghorbani, "Detecting malicious URLs using lexical analysis," in *Network and System Security*, J. Chen, V. Piuri, C. Su, and M. Yung, Eds. Berlin, Germany: Springer, 2016, pp. 467–482.
[8] M. Akiyama, T. Yagi, T. Yada, T. Mori, and Y. Kadobayashi, "Analyzing the ecosystem of malicious url redirection through longitudinal observation from honeypots," *Comput. Secur.*, vol. 69, pp. 155–173, 2017.
[9] K. Althobaiti, N. Meng, and K. Vaniea, "I don't need an expert! making URL phishing features human comprehensible," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2021, pp. 1–17.
[10] K. Althobaiti, G. Rummani, and K. Vaniea, "A review of human- and computer-facing url phishing features," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops*, 2019, pp. 182–191.
[11] T. T. T. Pham, V. N. Hoang, and T. N. Ha, "Exploring efficiency of character-level convolution neuron network and long short term memory on malicious URL detection," in *Proc. 7th Int. Conf. Netw. Commun. Comput.*, 2018, pp. 82–86.
[12] Y. Zhauniarovich, I. Khalil, T. Yu, and M. Dacier, "A survey on malicious domains detection through DNS data analysis," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–36, Jul. 2018.
[13] S. Rahman, T.-K. Huang, H. V. Madhyastha, and M. Faloutsos, "Detecting malicious facebook applications," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 773–787, Apr. 2016.
[14] V. Thambawita *et al.*, "An extensive study on cross-dataset bias and evaluation metrics interpretation for machine learning applied to gastrointestinal tract abnormality classification," *ACM Trans. Comput. Healthcare*, vol. 1, no. 3, pp. 1–29, Jun. 2020.
[15] A. AlEroud and G. Karabatis, "Bypassing detection of URL-based phishing attacks using generative adversarial deep neural networks," in *Proc. 6th Int. Workshop Secur. Privacy Anal.*, 2020, pp. 53–60.
[16] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Learning to detect malicious URLs," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 1–24, May 2011.
[17] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 3645–3650.
[18] L. Jiang, D. Meng, S.-I. Yu, Z. Lan, S. Shan, and A. Hauptmann, "Self-paced learning with diversity," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2078–2086.
[19] Y. Chen, F. M. Zahedi, A. Abbasi, and D. G. Dobolyi "Trust calibration of automated security IT artifacts: A multi-domain study of phishing-website detection tools," *Inf. Manage.*, vol. 58, no. 1, 2021, Art. no. 103394.
[20] H. Choi, B. Zhu, and H. Lee, "Detecting malicious web links and identifying their attack types," in *Proc. 2nd USENIX Conf. Web Appl. Develop.*, 2011, pp. 125–136, Art. no. 11.
[21] E. Baykan, M. Henzinger, L. Marian, and I. Weber, "Purely URL-based topic classification," in *Proc. 18th Int. Conf. World Wide Web*, 2009, pp. 1109–1110.
[22] R. Verma and A. Das, "What's in a URL: Fast feature extraction and malicious URL detection," in *Proc. 3rd ACM Int. Workshop Secur. Privacy Anal.*, 2017, pp. 55–63.
[23] C. Dai, X. Liu, and J. Lai, "Human action recognition using two-stream attention based LSTM networks," *Appl. Soft Comput.*, vol. 86, 2020, Art. no. 105820.
[24] Y. Liang, J. Kang, Z. Yu, B. Guo, X. Zheng, and S. He, "Leverage temporal convolutional network for the representation learning of urls," in *Proc. IEEE Int. Conf. Intell. Secur. Inform.*, 2019, pp. 74–79.
[25] X. Yan, Y. Xu, B. Cui, S. Zhang, T. Guo, and C. Li, "Learning URL embedding for malicious website detection," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6673–6681, Oct. 2020.
[26] M. Sameen, K. Han, and S. O. Hwang, "Phishhaven—An efficient real-time ai phishing URLs detection system," *IEEE Access*, vol. 8, pp. 83 425–83 443, 2020.
[27] H. Le, Q. Pham, D. Sahoo, and S. C. H. Hoi, "URLNet: Learning a url representation with deep learning for malicious url detection," 2018, *arXiv:1802.03162*.
[28] P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta, "PhishNet: Predictive blacklisting to detect phishing attacks," in *Proc. IEEE INFOCOM*, 2010, pp. 1–5.
[29] W. Yang, W. Zuo, and B. Cui, "Detecting malicious URLs via a keyword-based convolutional gated-recurrent-unit neural network," *IEEE Access*, vol. 7, pp. 29 891–29 900, 2019.
[30] S. Marchal, J. François, R. State, and T. Engel, "PhishStorm: Detecting phishing with streaming analytics," *IEEE Trans. Netw. Service Manag.*, vol. 11, no. 4, pp. 458–471, Dec. 2014.
[31] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin, "Evasion techniques: Sneaking through your intrusion detection/prevention systems," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 4, pp. 1011–1020, Fourth Quarter 2012.
[32] D. Li and Q. Li, "Adversarial deep ensemble: Evasion attacks and defenses for malware detection," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 3886–3900, 2020.
[33] M.-Y. Kan and H. O. N. Thi, "Fast webpage classification using URL features," in *Proc. 14th ACM Int. Conf. Inf. Knowl. Manage.*, 2005, pp. 325–326.
[34] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious web sites from suspicious URLs," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2009, pp. 1245–1254.
[35] P. Yang, G. Zhao, and P. Zeng, "Phishing website detection based on multidimensional features driven by deep learning," *IEEE Access*, vol. 7, pp. 15 196–15 209, 2019.
[36] S. C. H. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," *Neurocomputing*, vol. 459, pp. 249–289, 2021.
[37] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious URLs: An application of large-scale online learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 681–688.
[38] R. Rajalakshmi, J. Raymann, A. Prabu, and C. Aravindan, "Deep URL: Design of adult URL classifier using deep neural network," in *Proc. Int. Conf. Adv. Inf. Sci. Syst.*, 2019, pp. 1–5.
[39] M. Trevisan and I. Drago, "Robust URL classification with generative adversarial networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 46, no. 3, pp. 143–146, Jan. 2019.
[40] M. P. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *Proc. 23rd Int. Conf. Neural Inf. Process. Syst.*, 2010, pp. 1189–1197.
[41] H.-T. Cheng *et al.*, "Wide & deep learning for recommender systems," in *Proc. 1st Workshop Deep Learn. Recommender Syst.*, 2016, pp. 7–10.
[42] S. Rendle, "Factorization machines," in *Proc. IEEE Int. Conf. Data Mining*, 2010, pp. 995–1000.
[43] S. Deng, N. Zhang, W. Zhang, J. Chen, J. Z. Pan, and H. Chen, "Knowledge-driven stock trend prediction and explanation via temporal convolutional network," in *Proc. Companion World Wide Web Conf.*, 2019, pp. 678–685.
[44] L. Jiang, D. Meng, S.-I. Yu, Z. Lan, S. Shan, and A. G. Hauptmann, "Self-paced learning with diversity," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2078–2086.
[45] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.
[46] S. Prillo, "An elementary view on factorization machines," in *Proc. 11th ACM Conf. Recommender Syst.*, 2017, pp. 179–183.
[47] Y. Liang, X. Wang, Z. Yu, B. Guo, X. Zheng, and S. Samtani, "Energy-efficient collaborative sensing: Learning the latent correlations of heterogeneous sensors," *ACM Trans. Sensor Netw.*, vol. 17, no. 3, pp. 1–28, Jun. 2021.
[48] C. Wu, X.-J. Wu, and J. Kittler, "Spatial residual layer and dense connection block enhanced spatial temporal graph convolutional network for skeleton-based action recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop*, 2019, pp. 1740–1748.
[49] A. Zaeemzadeh, N. Rahnavard, and M. Shah, "Norm-preservation: Why residual networks can become extremely deep?," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 11, pp. 3980–3990, Nov. 2021.
[50] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018, *arXiv: Learning*. [Online]. Available: https://arxiv.org/abs/1803.01271?spm=a2c4e.11153940.blogcont642474.26.18fb6a59FjzYuR
[51] L. Lin, B. Xu, W. Wu, T. Richardson, and E. A. Bernal, "Medical time series classification with hierarchical attention-based temporal convolutional networks: A case study of myotonic dystrophy diagnosis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2019, pp. 83–86.
[52] S. A. Seyedi, S. S. Ghodsi, F. Akhlaghian, M. Jalili, and P. Moradi, "Self-paced multi-label learning with diversity," in *Proc. 11th Asian Conf. Mach. Learn.* 2019, pp. 790–805.

[53] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2014, pp. 1746–1751.
[54] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2267–2273.
[55] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proc. 15th Conf. Eur. Chapter Assoc. Comput. Linguistics*, 2017, pp. 427–431.
[56] R. Johnson and T. Zhang, "Deep pyramid convolutional neural networks for text categorization," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics*, 2017, pp. 562–570.
[57] A. Vaswani *et al.*, "Attention is all you need," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst., ser. NIPS'17.*, 2017, pp. 6000–6010. [Online]. Available: http://arxiv.org/abs/1706.03762

**Yunji Liang** received the PhD degree in computer science from Northwestern Polytechnical University, in 2016. He is currently an associate professor with Northwestern Polytechnical University, Xi'an, China. During 2012–2017, he worked with University of Arizona in USA as visiting scholar and postdoctoral researcher respectively. His research interests include pervasive computing, social computing and natural language processing.

**Qiushi Wang** is currently working toward the master's degree at the School of Computer Science, Northwestern Polytechnical University, China. His research interests include natural language processing and management of information system.

**Kang Xiong** is currently working toward the bachelor's degree at the School of Computer Science, Northwestern Polytechnical University, China. His research interests include machine learning and natural language processing.

**Xiaolong Zheng** is currently a professor with the State Key Laboratory of Management and Control for Complex System, Chinese Academy of Sciences Institute of Automation. His research interests include social media analytics and intelligence, social computing, and data mining.

**Zhiwen Yu** (Senior Member, IEEE) received the PhD degree in computer science and technology from Northwestern Polytechnical University. He is currently a professor with the School of Computer Science, Northwestern Polytechnical University, China. His research interests include cover pervasive computing, context-aware systems, human-computer interaction, and personalization.

**Daniel Zeng** (Fellow, IEEE) is currently a professor with the State Key Laboratory of Management and Control for Complex Systems, Chinese Academy of Sciences Institute of Automation. His research interests include multi-agent systems, collaborative information and knowledge management, recommender systems, social media analytics and intelligence, information, and security informatics.