Research Article

Detecting Multielement Algorithmically Generated Domain Names Based on Adaptive Embedding Model

Luhui Yang,1 Guangjie Liu,2 Weiwei Liu,1 Huiwen Bai,1 Jiangtao Zhai,3 and Yuewei Dai2

1School of Automation, Nanjing University of Science and Technology, Nanjing, China
2School of Electronic and Information Engineering, Nanjing University of Science and Technology, Nanjing, China
3School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, China

Correspondence should be addressed to Guangjie Liu; gjieliu@gmail.com

Received 19 January 2021; Revised 25 April 2021; Accepted 24 May 2021; Published 1 June 2021

Copyright © 2021 Luhui Yang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the development of detection algorithms on malicious dynamic domain names, domain generation algorithms have developed to be more stealthy. The use of multiple elements for generating domains will lead to higher detection difficulty. To effectively improve the detection accuracy of algorithmically generated domain names based on multiple elements, a domain name syntax model is proposed, which analyzes the multiple elements in domain names and their syntactic relationship, and an adaptive embedding method is proposed to achieve effective element parsing of domain names. A parallel convolutional model based on the feature selection module combined with an improved dynamic loss function based on curriculum learning is proposed, which can achieve effective detection on multielement malicious domain names. A series of experiments are designed and the proposed model is compared with five previous algorithms. The experimental results denote that the detection accuracy of the proposed model for multiple-element malicious domain names is significantly higher than that of the comparison algorithms and also has good adaptability to other types of malicious domain names.

1. Introduction

Advanced Persistent Threat (APT) attacks and botnets have become important threats to network security [1, 2]. To achieve remote control of the controlled hosts, attackers usually use a remote Command and Control (C & C) server. After establishing a communication link with the C & C server, the attacker can collect information about the controlled host and steal sensitive data or use the controlled host to launch further attacks on other hosts or networks. Early botnets would hard-code the domain names of C & C servers in malware, which can be easily detected and blocked by firewalls or Intrusion Detection Systems (IDS). To enable dynamic updating of domain names, Domain Generation Algorithm (DGA) is widely used in malware communication with C & C servers. Cyber attackers use DGAs to generate and register a large number of pseudorandom domain names and then embed the domain generation code in the malware that infiltrates into the hosts. When communicating with the C & C server, the algorithm is used to generate domain names to complete the communication. Because the domain names are usually dynamically generated, the malware’s communication domain names are updated over time, thus evading the detection rules of the security software. With the development of mobile Internet, 5G technology, and the increase of the Internet of Things (IoT) devices, detecting malicious algorithmically generated domain names is of great importance for network security.

DGAs can be classified into various categories according to their generation principles and generating elements [3]. Early DGAs used English letters, numbers, and some special characters as basic elements to generate domain names by some simple random algorithms. Then some botnets started to use words as the basic elements for domain generation, such as Matsnu [4] and Suppobox [5]. Compared with character-based DGAs, word-based DGAs are more difficult
to detect. Later, some researchers have designed DGAs that can generate domain names with lower character randomness by using combinations of English letters [6]. In recent years, with the development of detection techniques for malicious domain names, some researchers have designed DGAs that are more resistant to detection, such as a Stealthy Domain Generation Algorithm (SDGA) based on the Hidden Markov Model (HMM) [7] and the use of Generative Adversarial Networks GAN (GAN) to generate dynamic domain names [8]. According to the analysis of legitimate domains, there are elements such as English words, Chinese pinyin, abbreviations, special meaning numbers, English letters, and special characters in legitimate domain names, so when the domain names are generated based on these multiple elements, the degree of anomalies in the grammatical composition is lower and the existing algorithms are not effective for the detection of malicious domain names based on multiple elements.

To effectively improve the detection accuracy of multi-elements hybrid malicious domain names, a domain name multielement adaptive embedding method based on a domain name syntax model is proposed, which can realize the effective segmentation of various elements in domain names. Based on the adaptive embedding module, a parallel convolutional model based on the feature selection module is proposed, which, combined with a dynamic Focalloss function, can achieve effective feature extraction and classification of multielement hybrid malicious domain names. Comparative experimental results on several typical datasets denote that the proposed model can effectively detect multielement hybrid malicious domain names and also has good adaptability to other types of malicious domain names.

This paper’s contributions can be summarized as follows:

(1) We propose a domain name syntax model, which analyzes the constituent elements and syntax rules of domain names. It can be used for the element parsing of domain names.

(2) We propose an adaptive parsing and embedding module, which can get more appropriate domain name parsing results based on the maximum entropy probability. It can map the elements into vector representations.

(3) We propose a parallel convolutional model based on a feature selection module to improve the classification accuracy by performing feature attention on multiple convolutional branches.

(4) We proposed an improved dynamic loss function based on curriculum learning, which can effectively improve the detection accuracy for hard-to-detect samples.

The remainder of this paper is organized as follows. In the next section, we summarize the related work on the existing malicious domain name detection methods. In Section 3, the proposed domain name syntax model is introduced. In Section 4, the proposed detection model is introduced. In Section 5, we firstly introduce the selected dataset in our research and then a series of experiments are conducted on the datasets. Finally, in Section 6, we conclude our research and future work.

2. Related Work

When malware communicates with C & C servers, the requested domain name is usually a dynamic domain name generated by DGA. Detecting malicious dynamic domain names can discover the communication behaviors of malware and C & C nodes. Methods for detecting malicious DNS communication can be divided into three categories based on their principles: feature-based, behavior-based, and deep-learning-based. Among them, feature-based detection algorithms analyze the character features of malicious domain names and legitimate domain names. The behavior-based algorithms analyze the behavioral features of malicious and legitimate DNS communication. Deep-learning-based algorithms use deep-learning-based models to extract features, which do not rely on manual analysis of features and have better self-learning and self-adaptive capabilities and can detect unknown malicious domain names. Deep-learning-based algorithms are the most popular methods in recent years, so they are grouped into a separate category. The following is a review of the three types of detection algorithms.

Domain name feature-based detection algorithms are the earliest and classical malicious domain name detection algorithms, which are implemented by analyzing the character features of domain names. This type of detection algorithms only relies on the analysis of domain name features without long-time observation and additional information. Yadav et al. [9, 10] detected malicious domain names by extracting the Kullback-Leibler Divergence, Jaccard Index, and Edit Distance features; they also use a linear regression classifier to achieve effective detection. However, the algorithm was tested only on a few botnet families, not on a larger DGA dataset. Later, Schiavoni et al. [11] proposed Phoenix, a domain name detection algorithm, which uses a meaningful character ratio and N-gram normality as features. The shortcomings of this algorithm lie in the small number of feature dimensions and the fact that it has been tested only on a limited number of malicious dynamic domain names. Raghuram et al. [12] focused on the readability of domain names; they analyzed the character composition, word composition, word length, and other dimensions of legitimate domain names and established a probability model of legitimate domain names from a large number of samples to achieve a better effect of detecting malicious domain names. Truong et al. [13] analyzed domain name construction rules based on a large number of legitimate and malicious domain names to detect malicious domain names by the length expectation value. Schales et al. [14] performed detection by extracting 17 domain name features combined with four of weighted confidence and verified the effectiveness of the algorithm in a large-scale network environment. Yang et al. [15] proposed a word semantic analysis method for word-based domain names to detect malicious domain names by the correlation between words in them.
In recent years, deep-learning-based DGA domain name detection has become the most popular direction for malicious domain name detection. Woodbridge et al. [23] first used a deep learning model to detect malicious domain names, using Long Short Term Memory (LSTM) network to design a detection model. Yu et al. [24] built a model for detecting malicious domain names using CNNs and achieved detection results close to the LSTM model. Subsequently, Yu et al. [25] compared the detection results of seven classical deep-learning-based models and analyzed the detection effects of different CNN and LSTM architectures; the results denote that both LSTM and CNN structures had effective detection results of different CNN and LSTM architectures; the results denote that they rely on manual feature analysis capabilities, and the dimensionality of character-level features is limited. After verifying the effectiveness of domain name features, Zago et al. [16] found that only a few features had a significant impact on the detection results. The malware communicates with C & C servers with similar lifecycle and query patterns in their DNS requests. Therefore, some researchers have combined the communicational behavior features of DNS with character-level features of the domain name to designing detection algorithms. Bilge et al. [17] summarized 15 features of domain name requests based on the features analyzed and designed a malicious domain name detection system called EXPOSURE using a J-48 decision tree as the classifier. The system successfully detected a large number of malicious domain names in real network traffic. Shi et al. [18] extracted domain name features, IP features, TTL features, and WHOIS features and used Extreme Learning Machine (ELM) to achieve effective classification. Mowbray et al. [19] proposed a malicious domain name detection algorithm based on domain name length distribution, which can achieve effective detection of some specific kinds of malicious domain names. Kwon et al. [20] used the Power Spectral Density (PSD) analysis technique to analyze DNS query time information in large-scale networks to detect malicious DNS requests. Since the generation and registration of DGA domain names are not fully synchronized, a large number of unregistered domain names appear in the DNS requests of malware; based on this fact, Yadav et al. [21] and Antonakakis et al. [22] have proposed various malicious domain name request detection algorithms based on NXDomain responses caused by unregistered domain names.

In recent years, deep-learning-based DGA domain name detection has become the most popular direction for malicious domain name detection. Woodbridge et al. [23] first used a deep learning model to detect malicious domain names, using Long Short Term Memory (LSTM) network to design a detection model. Yu et al. [24] built a model for detecting malicious domain names using CNNs and achieved detection results close to the LSTM model. Subsequently, Yu et al. [25] compared the detection results of seven classical deep-learning-based models and analyzed the detection effects of different CNN and LSTM architectures; the results denote that both LSTM and CNN structures had effective detection results. Tran et al. [26] proposed an LSTM-MI model, which can effectively alleviate the sample imbalance in the binary classification and multiclassification problems by using a cost-sensitive loss function. Also to improve the accuracy of the multiclassification problem, Qiao et al. [27] added a global attention mechanism to the LSTM model, which can significantly improve the multiclassification accuracy of DGA domain names. The global attention mechanism relies on the calculation of the global attention matrix, which is strongly influenced by different training samples, and the local attention mechanism can be considered to improve the attention effect by calculating only the weights between different characters in the domain name. Therefore, Yang et al. [28] proposed a detection model combining CNN and LSTM with local attention mechanism, which can effectively improve the detection accuracy of some hard-to-detect malicious domain names. Xu et al. [29] improved the embedding method of domain name by N-gram characters and used a parallel CNN network for classification.

DGA algorithms have also evolved in recent years to generate stealthier malicious domain names. For example, the malware Matsnu and Suppobox use English words to generate domain names; it can effectively reduce the randomness of characters in domain names. On the other hand, Fu et al. [7] proposed a stealthy domain name generation algorithm, which is based on Hidden Markov Models (HMM) and Probabilistic Context-Free Grammars (PCFGs). The generative model is designed and trained with a large number of legitimate samples that generate domain names that better fit the character distribution of legitimate domain names. Therefore, more effective detection models need to be designed for stealthier malicious domain names.

In summary, compared with the domain name feature-based detection algorithm, the deep-learning-based detection algorithm does not need to manually analyze the features and extracts higher-dimensional domain name features through the deep-learning-based architectures. Compared with behavioral feature-based detection algorithms, deep-learning-based detection algorithms do not need to collect the complete DNS interaction. However, the embedding method of current deep-learning-based algorithms is relatively simple, and most of them only use character embedding, which cannot effectively represent domain names with multiple elements, so it is necessary to analyze different elements in domain names and design more effective embedding methods.

3. Domain Name Syntax Model

This section defines a domain name syntax model. Firstly, the constituent elements of domain names are analyzed, and the basic constituent units of the domain name syntax tree are defined from multiple dimensions. Then, the multidimensional tree structures of the domain name are analyzed. Finally, the features of legitimate domain names and malicious dynamic domain names are compared based on the syntax model, and the results denote that the domain name syntax tree can effectively characterize the domain name and can support the design of the detection model of malicious domain names.

3.1. Elements Definition of Domain Name Syntax Model

The basic elements of a domain name are numbers, English letters, and special characters. The numbers are the 10 digits from 0 to 9, and the English letters are usually lowercase from English alphabets. The special characters in legitimate domain names usually only contain ‘-’ and ‘_’ according to the statistical result of the Alexa domain list. There are various combinations of English letters in domain names that can form elements with certain semantic meanings, such as English words, Chinese pinyin, abbreviations, and so
on. Different numbers can also form numeric combinations with special meanings, such as years. To analyze the elements in domain names effectively, the constituent elements in domain names are defined in multiple dimensions.

Firstly, the basic elements of the domain name are defined in equation (1); the basic elements are 10 numbers, 26 English letters, and 2 special characters.

\[
\begin{align*}
\hat{A} &= \{a-z\}, \\
\hat{N} &= \{0-9\}, \\
\hat{T} &= \{', '-', 'J\},
\end{align*}
\]  

(1)

where \(\hat{A}\) denotes the set of English letters, \(\hat{N}\) denotes the set of numbers, and \(\hat{T}\) denotes the set of special characters.

There are a large number of English words in the legitimate domain names, which can be regarded as one type of element in domain names. It can be defined as

\[ W = \{w_1, w_2, \ldots, w_n\}, \]  

(2)

where \(W\) denotes the set of English words.

The total number of two-character combinations in the domain name is 676, and the top 50 two-character combinations can be taken as the meaningful two-character set. The total number of three-character combinations in the domain name is 17,576, and the top 100 three-character combinations can be taken as the meaningful three-character set. So the meaningful two-character and three-character sets in the domain name are defined as

\[
\begin{align*}
B &= \{b_1, b_2, \ldots, b_{50}\}, \\
F &= \{f_1, f_2, \ldots, f_{100}\},
\end{align*}
\]  

(3)

where \(B\) denotes the set of meaningful two-character combinations, and \(F\) denotes the set of meaningful three-character combinations.

Other elements in the domain name include Chinese pinyin, English abbreviations, Chinese abbreviations, and special meaning numeric combinations, which can be defined as

\[
\begin{align*}
\hat{G} &= \{g_1, g_2, \ldots, g_n\}, \\
\hat{H} &= \{h_1, h_2, \ldots, h_n\}, \\
\hat{I} &= \{i_1, i_2, \ldots, i_n\}, \\
\hat{J} &= \{j_1, j_2, \ldots, j_n\},
\end{align*}
\]  

(4)

where \(\hat{G}\) denotes the set of Chinese pinyin, \(\hat{H}\) denotes the set of English abbreviations, \(\hat{I}\) denotes the set of Chinese abbreviations, and \(\hat{J}\) denotes the set of meaning numeric combinations.

3.2. Domain Syntax Tree Construction Methods. Tree structures can be constructed in several dimensions for describing the relationship between elements in a domain name or between different domain names. In this paper, we focus on the syntactic composition of the relationships between different elements in a domain name. From the analysis in the previous subsection, it is clear that the same domain name can be divided into various types of elements, and the characters in a domain name can be divided according to single, double, or triple characters, as well as the pinyin, word, and number combinations. Take the domain name "baidutop123" as an example. The domain name can be parsed according to single, double, or triple characters; however, they cannot obtain the semantic meaning of the characters in the domain name; the most appropriate parsing method that satisfies human understanding should parse the domain to "bai," "du," "top," "123".

To obtain more appropriate domain name parsing results, a domain name syntax model based on PCFG is proposed. The PCFG is a syntax model based on word types and cooccurrence probabilities that parses sentences into tree-like element parse trees. The PCFG model only considers words, while the elements of domain names are more complex. According to the analysis in the previous subsection, the set of elements in a domain name include English letters, numbers, special characters, words, two-character combinations, three-character combinations, Chinese pinyin, Chinese abbreviations, English abbreviations, and special meaning numerical combinations. The combination of different elements can be defined as

\[ WN \rightarrow W \circ N, \]  

(5)

where \(W\) denotes a word, and \(N\) denotes a number. It can be further expressed as

\[
\begin{align*}
W &\rightarrow \text{top}, \\
N &\rightarrow 1.
\end{align*}
\]  

(6)

Therefore, the parse tree of "baidutop123" can be obtained based on the domain name PCFG as shown in Figure 1.

In Figure 1, the symbol \(D\) denotes the domain name, the individual branches denote the grammatical composition in the domain name, the symbols in equation (5) denote the nonterminal symbol of the syntax tree, and the symbol in equation (6) denotes the terminal symbol.

Since the elements in the domain names are not randomly combined, there is a certain combination probability, so the PCFG model enhances the rules by adding a probability to each rule:

\[ A \rightarrow \beta[p], \]  

(7)

where \(A\) denotes the nonterminal symbols, \(\beta\) denotes the concatenation of all terminal symbols and nonterminal symbols, and \(p\) denotes the probability of occurrence of the rule. The domain name syntax model can be defined as shown in equation (8).

\[ G = (M, \Sigma, R, D, P), \]  

(8)

where the conditional probability of a nonterminal symbol \(A\) connecting a symbol can be expressed as \(P(A \rightarrow \beta[A])\).

The words in a domain name can also be parsed into characters or two-character combinations, etc. Therefore, there are different priorities for parsing the characters into different elements. Segmenting the elements in a domain
name according to higher priorities can yield a more accurate domain name parse tree. After adding the domain name resolution priority, the domain name syntax model can be defined as

\[ G = (M, \Sigma, R, D, P, L), \]

where \( L \) indicates the priority of the elements in each domain parsing rule.

For a grammar parse tree \( T \) obtained from a domain name \( D \), the average priority of elements in each rule and the probability of this parse tree can be calculated as

\[ \begin{align*}
L(T, D) &= \frac{1}{n} \sum l(r(n)), \\
P(T, D) &= \prod_{n \in T} p(r(n)),
\end{align*} \]

where \( L(T, D) \) denotes the average priority of each domain name parsing result, \( P(T, D) \) denotes the conditional probability of that parsing result, \( l(r(n)) \) denotes the priority of each element in the parsing rule, and \( p(r(n)) \) denotes the conditional probability of each parsing rule.

For multiple parse trees of the same domain name, the one with the highest average priority and the highest probability can be selected as

\[ \bar{\mathbf{T}}(D) = \arg\max_{T \in \mathbf{T}(D)} L(T, D) \& \arg\max_{T \in \mathbf{T}(D)} P(T, D), \]

where \( \mathbf{T}(D) \) denotes the set of all parse trees for domain \( D \).

The conditional probability of each rule can be derived from the statistical results of a large number of samples and can be calculated as

\[ P(A \to \beta | A) = \frac{C(A \to \beta)}{C(A)}, \]

where \( C(A \to \beta) \) denotes the number of occurrences of \( A \to \beta \) in the statistical samples and \( C(A) \) denotes the number of occurrences of the symbol \( A \) in the statistical samples.

While there exist multiple dimensional ways to analyze the relationship of elements in a domain name, this section focuses on constructing the syntax tree of a domain name from the element type and the conditional probability of occurrence. By analyzing the relationship between the elements in a domain name, the syntax tree of the elements can be constructed, and the domain name legitimacy can be analyzed based on the domain name syntax tree.

### 3.3. Analysis of Malicious Domain Names

Currently known DGAs can be divided into three categories. The first uses characters as generating elements to randomly generate domain names; most of the current malware uses these algorithms. The second uses English words to generate domain names, reducing the randomness of the characters in domain names; a small number of current malware uses these algorithms. The third is the SDGA algorithm based on HMM and other generative models; the HMM models generate domain names with character distribution closer to normal domain names. This section analyzes four typical samples of malware-generated domain names based on random character DGAs, in which Corebot and Gameover generate malicious dynamic domain names using 26 English letters and 10 numbers; Cryptolocker and Kraken generate malicious dynamic domain names using only 26 English letters. In the word-based DGA, Matsnu randomly selects some verbs and nouns from the word list and uses "_" to connect them, and Suppobox selects two words from the word list to form a dynamic domain name. The HMM-based SDGA algorithm uses English words or legitimate domain names as training samples to train the HMM model and uses the pretrained model to generate malicious dynamic domain names. Sample examples of the three DGA algorithms are shown in Table 1.

To analyze the character characteristics of different types of malicious domain names, the character frequency distribution for legitimate domain names is counted, and three types of malicious domain names are selected for statistics. The results of character frequency distribution are shown in Figure 2. As can be seen from the figure, the character frequency of character-based malicious domain names differs significantly from that of legitimate domain names, while the character frequency distribution of word-based and SDGA domain names is closer to that of legitimate domain names, indicating that word-based constructed malicious domain names and SDGA domain names are less abnormal in character distribution and more concealed.

To further analyze the differences of several malicious domain names, the entropy value of domain name parsing results and distributions of entropy values are counted. For the domain name parsing result \( D = \{ c_1, c_2, \ldots, c_n \} \), the entropy can be calculated as

\[ H(c_1, c_2, \ldots, c_n) = \sum_{k=1}^{n} P(c_k | c_1 \ldots c_{k-1}) \log_2 P(c_k | c_1 \ldots c_{k-1}). \]

---

**Figure 1:** Illustration of domain name PCFG syntax parse tree.
The mean entropy of Matsnu domain names is significantly lower than that of legitimate domains, mainly because the words in Matsnu are connected by the character "—", so the average probability is significantly lower. The average entropy of Suppobox domain names, which is composed entirely of words, is closer to that of the legitimate domain name; however, the peak of the histogram distribution is significantly different from the legitimate domain names.

For SDGA domain names, 10,000 legitimate domain names, 10,000 SDGA domain names generated by training with legitimate domain names, and 10,000 SDGA domain names generated by training with English words are selected and their average entropy distribution histograms are as shown in Figure 5. It can be seen that since the SDGA domain names are trained with legitimate domain names or English words, the generated domain names are closer in distribution to legitimate domain names.

The results of parsing and statistical analysis based on domain name syntax tree show that many types of malicious domain names have a certain degree of difference in entropy distribution from legitimate domain names; however, some word-based and SDGA domain names are closer to legitimate domain names in entropy distribution, indicating that such malicious domain names are more concealed, and the detection method needs to deeply analyze the semantic expression of elements.

### Table 1: DGA categories and examples.

| DGA category     | Malware Samples                | Word-based DGA | HMM-based SDGA |
|------------------|--------------------------------|----------------|----------------|
| Corebot          | hkf1tw674cfcp32mjy.ddns.net    | text-react-teacher.com | stgaars.net    |
| Cryptolocker     | 7pg2ir32y8vbg.ddns.net         | passenger-express.com | dap6ann.net    |
| Gameover         | g4afw758tqna.ddns.net          | wall-turn-panic.com | pore9nn10.net  |
| Kraken           | qydkyiugnjio.net               | partyprobable.net |               |
| matsnu           | upepekvlmmtwag.ru              | fightprobable.net |               |
| Suppobox         | 1weas0akbztv6czy30bwhl01terk.net | freshwelcome.net |               |

The mean entropy value is calculated as

\[
\frac{1}{n} H(c_1, c_2, \ldots, c_n) = -\frac{1}{n} \sum_{k=1}^{n} P(c_k|c_1 \ldots c_{k-1}) \log_2 P(c_k|c_1 \ldots c_{k-1}),
\]

\[(14)\]

where \( n \) denotes the number of elements in a domain name.

Based on the Markov Assumption, the probability of occurrence of each element in the domain name is only related to the former \( N - 1 \) characters. For the 2-gram model, the average entropy of the domain name is calculated as

\[
\frac{1}{n} H(c_1, c_2, \ldots, c_n) = -\frac{1}{n} \sum_{k=1}^{n} P(c_k|c_{k-1}) \log_2 P(c_k|c_{k-1}),
\]

\[(15)\]

and for character-based domain names, 10,000 legitimate samples, Corebot samples, and Cryptolocker samples are selected and their average entropy distribution histograms are calculated as shown in Figure 3. Since the occurrence of each character is more random in Corebot and Cryptolocker, their conditional probability of character occurrence is lower compared with that of legitimate domain names, and thus the average entropy is significantly lower than that of legitimate domain names.

For word-based domains, 10,000 legitimate domain names, 10,000 Matsnu domain names, and 10,000 Suppobox domain names are selected and the histograms of their domain name mean entropy distribution are as shown in Figure 4.

The mean entropy of Matsnu domain names is significantly lower than that of legitimate domains, mainly because the words in Matsnu are connected by the character "—", so the average probability is significantly lower. The average entropy of Suppobox domain names, which is composed entirely of words, is closer to that of the legitimate domain name; however, the peak of the histogram distribution is significantly different from the legitimate domain names.

4. Detection Model Based on Adaptive Embedding

Based on the domain name syntax tree model, this section firstly proposes an adaptive domain name element embedding method to map different elements to vectors, then a parallel convolutional model based on a feature selection module is proposed to select features from different convolutional kernel branches to improve the accuracy of feature extraction, and finally, a dynamic loss function based on curriculum learning is proposed to improve the training effect of the model.

#### 4.1. Adaptive Embedding Module

In most previous dynamic domain name detection models, the embedding is based on each character in the domain name, and the relationship between characters is not processed. The literature [29] proposed an N-gram embedding method, which can improve the perception of the association between different characters in the domain name. It is clear from the analysis in Section 3 that there are multiple types of elements in domain names, and the N-gram embedding cannot effectively express the different elements in domain names. In this research, we believe that a more accurate segmentation of different elements in domain names, along with appropriate embedding, can improve the accuracy of the representation of domain names and thus improve the detection effect.
Figure 2: Frequency statistics of character distribution of legitimate and four malicious domain names.

Figure 3: Histogram of the average entropy distribution of legitimate, Corebot, and Cryptolocker domain names.
Figure 4: Histogram of the average entropy distribution of legitimate, Matsnu, and Suppobox domain names.

Figure 5: Continued.
The domain name element vector is generated as shown in equation (16).

\[
\hat{A} = \{a - z\}, \quad \hat{N} = \{0 - 9\}, \quad \hat{T} = \{\cdot, \ldots, \cdot\}, \\
\hat{W} = \{w_1, w_2, \ldots, w_n\}, \quad \hat{B} = \{b_1, b_2, \ldots, b_{50}\}, \\
\hat{F} = \{f_1, f_2, \ldots, f_{100}\}, \quad \hat{G} = \{g_1, g_2, \ldots, g_{70}\}, \\
\hat{H} = \{h_1, h_2, \ldots, h_n\}, \quad \hat{I} = \{i_1, i_2, \ldots, i_n\}, \\
\hat{J} = \{j_1, j_2, \ldots, j_n\}, \quad \hat{E} = \{\hat{A}, \hat{N}, \hat{T}, \hat{W}, \hat{B}, \hat{F}, \hat{G}, \hat{H}, \hat{I}, \hat{J}\},
\]

where \(\hat{E}\) denotes the set of all elements in the domain names, \(\hat{A}\) denotes English letters, \(\hat{N}\) denotes numbers, \(\hat{T}\) denotes special characters, \(\hat{W}\) denotes English words, \(\hat{B}\) denotes two-character combinations, \(\hat{F}\) denotes three-character combinations, \(\hat{G}\) denotes Chinese pinyin, \(\hat{H}\) denotes English abbreviations, \(\hat{I}\) denotes Chinese abbreviations, and \(\hat{J}\) denotes numeric combinations.

The different elements are not completely unrelated to each other, for example, English words contain characters and two-character combinations and three-character combinations, and numeric combinations contain numbers. Therefore, for dividing strings into different elements in a domain name, it is necessary to define the priority level and divide them according to the priority level from highest to lowest. The definition of priority is shown in Table 2, and the larger the priority value the lower the priority. Among them, English words, Chinese pinyin, English abbreviations, Chinese abbreviations, and numeric combinations have the highest priority. Two-character combinations and three-character combinations have lower priority. The rest of elements have the lowest priority, which means the domain name will be split into single-character elements, including English letters, numbers, and special characters, only when it cannot be split into other elements.

A domain name can be parsed to different trees, each of which contains different element types. For different parse trees, the average priority of each tree is calculated, and the result with the highest priority is taken. For the parse trees with the same priority, by calculating the average entropy of each tree, the result with the highest entropy value is taken as the final domain name parsing result. The algorithm is as follows (Algorithm 1).

The segmentation method used in step 1 is the bidirectional maximum matching algorithm [15]. A unique parsing result \(T^*_e = \{e_1, e_2, \ldots, e_n\}\) can be obtained through the domain name syntax parsing algorithm. For each element, a vector can be obtained using Word2vec [30]. To obtain a more accurate word representation, a pretrained word vector model is used to embed the words. For other elements in the domain name, the collected domain names are used as a corpus to parse the elements in the domain name and the elements are trained with Word2vec to obtain the embedding model, and the length of each element vector is 128. Since there are many element types in domain names, different combinations of element types in domain names have different impacts on the legitimacy determination. To obtain more accurate domain name element vectors, the types of domain names are also mapped to vectors and are added to the element vector. Domain names are divided into 10 major categories: English letters, numbers, special characters, English words, Chinese pinyin, English abbreviations, Chinese abbreviations, numeric combinations, double-character combinations, and three-character combinations. Some element types can be subdivided into more subcategories. English letters can be divided into two categories: vowels and consonants, and English words can be divided into various lexical types according to the classification method of natural language processing. Each element type in the domain name is mapped to a vector of length 128, which is stitched with the original element vector to a vector of length 256. The vector length 128 is an experimentally obtained value that can balance computational efficiency and accuracy and is also used in the model of [28]. The domain name element vector is generated as shown in equation (17).

\[
V(e_k) = [\text{Embedding}(e_k), \text{Embedding}(X(e_k))],
\]

where \(V(e_k)\) denotes the vector of elements in the domain name and \(X(e_k)\) denotes the element type.

Domain name element parsing and embedding process are shown in Figure 6. Take the domain name “baidu.top1” for example, which contains Chinese pinyin, English words, and numbers; the most appropriate way is to parse it into the elements “bai,” “du,” “top,” and “1” and then map each resolved element and its type to a vector. In this case, the
English word “top” is mapped using a public pretrained embedding model, while “bai” and “du” and “l” are obtained by training elements from the collected corpus.

4.2. Parallel Convolutional Model Based on a Feature Selection Module. After embedding the elements in the domain name, a parallel convolutional model based on the feature selection module is designed to achieve effective classification. The feature selection module is combined with a multibranch parallel convolutional structure to achieve more effective feature extraction. Several previous works have verified the effectiveness of CNN-based models in the domain name detection field. Convolutional layers are mainly used to obtain the intrinsic association of adjacent elements by convolutional computation. Convolutional layers with different sizes of convolutional kernels are used to obtain the association between adjacent elements of the different perceived ranges. In the domain name detection problem, the elements are usually mapped to one-dimensional vectors, and then the one-dimensional vectors are subjected to one-dimensional convolution. For two adjacent element vectors $V_1 = [v_{11}, v_{12}, \ldots, v_{1n}]$ and $V_2 = [v_{21}, v_{22}, \ldots, v_{2m}]$, the convolution is calculated as shown in equation (18).

$$S(n) = \sum_{m=0}^{n-1} v_{1m} \cdot v_{2(n-m)}.$$  \hspace{1cm} (18)

In convolutional neural networks, convolution parameters and bias terms are usually added to the convolutional computation, and different convolutional kernel sizes are used, so the convolution process can be represented by

$$\begin{align*}
  c_i &= f(w \cdot V + b) \\
  V &= [V_i, V_{i+1}, \ldots, V_{i+k}],
\end{align*}$$  \hspace{1cm} (19)

where $c_i$ denotes the result of the convolution output, $f$ denotes the nonlinear activation function, $w$ denotes the parameters of the convolution, $b$ denotes the bias term, $V$ denotes the element vector of a window, and $h$ denotes the convolution window size, i.e., the convolution kernel size.

To extract convolutional features in different perceptual ranges, four convolutional kernels with different sizes are used to convolve the input element vectors, and the convolutional kernel sizes are taken as 2, 3, 5, and 7. In a domain name, the correlation ranges of elements are different, and the effective perceptual fields vary in different domain names. To improve the selection validity of the convolution results for different convolution kernel sizes, a feature selection module for one-dimensional parallel convolution is proposed, which is similar to SKNets [31]. The architecture of the model is shown in Figure 7.

In the model, domain name element vectors are firstly obtained by adaptive embedding. Then four convolutional layers with different kernel sizes are used to convolve the element vectors, respectively, which can obtain four convolutional feature maps with different perceptual ranges, and the attention weights of the four feature maps are calculated by the feature selection module. After that, the original feature maps are multiplied with the weight results, and finally, the four feature maps are summed to obtain the extracted domain name features. The extracted feature maps are input into three fully connected layers to obtain the binary classification results.

In the feature selection module, four feature maps are elementwise summed to obtain a feature map that fully mixes the four perceptual results:

$$U = U_1 + U_2 + U_3 + U_4.$$  \hspace{1cm} (20)

The global average pooling layer is then used to obtain the global information of each filter channel. For the $c$-th filter channel, if the number of elements is $N$, the global average pooling result is calculated as

$$S_c = \frac{1}{N} \sum_{i=1}^{N} U_c(i).$$  \hspace{1cm} (21)

To obtain feature weights adaptively, the results of the global average pooling are calculated using a fully connected layer:

$$Z = \delta(WS),$$  \hspace{1cm} (22)

where $\delta$ denotes the ReLU activation function and $W$ denotes the fully connected parameter.

To implement the weight calculation for different convolutional branch channels, a soft attention model is used and the normalized weight results are obtained using Softmax. The weight calculation for the four branches is

$$\begin{align*}
  a_c &= \frac{e^{A, Z}}{e^{A, Z} + e^{B, Z} + e^{C, Z} + e^{D, Z}}, \\
  b_c &= \frac{e^{B, Z}}{e^{A, Z} + e^{B, Z} + e^{C, Z} + e^{D, Z}}, \\
  c_c &= \frac{e^{C, Z}}{e^{A, Z} + e^{B, Z} + e^{C, Z} + e^{D, Z}}, \\
  d_c &= \frac{e^{D, Z}}{e^{A, Z} + e^{B, Z} + e^{C, Z} + e^{D, Z}}.
\end{align*}$$  \hspace{1cm} (23)
where $a$, $b$, $c$, and $d$ denote the attention weight vectors of $U_1$ to $U_4$, respectively.

Based on the attention weights, the results of the four convolutional branches can be weighted, and the sum of the weights of the four convolutional branches is 1 to obtain the feature map result $V$. The calculation of every channel satisfies

$$V_c = a_c \cdot U_1 + b_c \cdot U_2 + c_c \cdot U_3 + d_c \cdot U_4, \quad a_c + b_c + c_c + d_c = 1.$$  

(24)

The feature selection module can adaptively select features extracted by different convolutional kernels by using a soft attention mechanism to calculate the weights of multiple convolutional branches, which can achieve adaptive selection of features extracted from different convolutional kernels.

**4.3. Improved Dynamic Loss Function.** To improve the detection effect of the model, the detection accuracy for hard-to-detect samples needs to be improved, for which an improved dynamic loss function is proposed based on the idea of curriculum learning and the Focal loss [32]. It can adaptively adjust the loss of hard-to-detect samples at different stages of training to improve the detection accuracy of hard-to-detect samples and thus improve the overall detection effect of the model.

In the binary classification task, the cross-entropy loss function is usually used for model training:

$$L = -y \log y' - (1 - y) \log (1 - y'),$$  

(25)

where $y'$ is the prediction result and $y$ is the ground truth; the ground truth is 0 or 1 in binary classification. When $y = 1$, the closer $y'$ to 1 the smaller the value of loss generated, which indicates the lower difficulty of sample detection. When $y = 0$, the closer $y'$ to 0 the smaller the loss value generated. Hard-to-detect samples can be defined as samples that produce large loss values, and easy-to-detect samples can be defined as samples that produce small loss values. Since the number of easy-to-detect samples is much more than the hard-to-detect samples, to reduce the weight
of the loss generated by the easy-to-detect samples and increase the loss weight of the hard-to-detect samples. FocalLoss introduces a weighting factor, which makes the model pay more attention to the hard-to-detect samples, and the FocalLoss is

\[ L_{fl} = \begin{cases} 1 - y)^t \log y', & y = 1, \\ y^t \log (1 - y'), & y = 0, \end{cases} \]

(26)

and the curves of the FocalLoss with different \( y \) are shown in Figure 8. It can be seen that as the value of \( y \) increases, the difference between the loss value generated by samples close to the ground truth and samples far from the ground truth increases significantly, the weight of the loss generated by easy-to-detect samples is reduced, and the weight of the loss of hard-to-detect samples is increased.

In the FocalLoss function, the larger the value of \( y \) the higher the weight of the loss of hard-to-detect samples. However, according to the idea of curriculum learning, the model training is similar to the characteristics of human learning which learn the course from easy to hard; it can make the model better optimized [33]. Therefore, in the early stage of the model training phase, it is not necessary to pay too much attention to hard-to-detect samples, and after the model achieves a better detection effect on easy-to-detect samples, the loss weight of hard-to-detect samples is gradually increased, which makes the model pay more attention to difficult detection samples. Based on the above considerations, when using the FocalLoss function, the \( y \) value can be dynamically adjusted at different stages of the model training phase to achieve dynamic adjustment of the loss weights of hard-to-detect and easy-to-detect samples. For this purpose, the definition of dynamic \( y \) value is given as

\[ y = \alpha \cdot t^\beta, \]

(27)

where \( t \) denotes the training epochs, and \( \alpha \) and \( \beta \) are the fixed parameters.

As shown in Figure 9, the value of \( y \) increases as the number of training epochs increases, and the range of variation is affected by \( \alpha \) and \( \beta \). The parameter \( \alpha \) should be adjusted according to the convergence of the model. In the model with more difficult convergence and more training epochs, smaller \( \alpha \) should be used, while in the model with fewer training epochs, larger \( \alpha \) should be used. The parameter \( \beta \) should be set according to the training samples, and a larger \( \beta \) should be used for models that require a larger differentiation weight between difficult and easy samples; otherwise, a smaller \( \beta \) should be used.

Further, to limit the value of \( y \), a truncation threshold range \( [y_1, y_2] \) for \( y \) can be set:

\[ \begin{align*}
  y &= 0, & y < y_1, \\
  y &= \alpha \cdot t^\beta, & y_1 \leq y \leq y_2, \\
  y &= y_2, & y > y_2.
\end{align*} \]

(28)

An improved FocalLoss function is proposed that can dynamically adjust the loss weights of hard-to-detect and easy-to-detect samples at different stages of the training phase, which can make the model focus on hard-to-detect samples with different weights at different stages, and improve the detection accuracy of hard-to-detect samples while maintaining the detection rate of easy-to-detect samples.

5. Experiments and Analysis

In this section, several experiments are designed to evaluate the model. Firstly, an ablation experiment is designed to verify the effectiveness of the proposed adaptive embedding module, feature selection module, and improved dynamic loss function, and the results denote that the proposed modules are effective in improving the detection accuracy of the model. Then the proposed detection algorithm is compared with several previous algorithms, and the results denote that the proposed algorithm can effectively improve detection accuracy for many types of malicious dynamic domain names.

5.1. Experimental Preparation. In this experiment, a large number of legitimate domain names and multiple types of malicious domain names are collected from public datasets. For the legitimate domain names, we collected the top 1 million domain names of Alexa’s daily updated rankings from January 2009 to February 2019, 3159 collected lists for total, and for each list, we selected the top 200,000 to add to the whitelist, and after cleaning and merging, a whitelist sample of over 10 million is obtained. For multiple malicious domain names with different elements, we selected 20 samples of DGAs containing typical elements and generation methods from the UMUDGA [34] dataset, with 10,000 of each DGA family. For SDGAs, 8 types of SDGA domain names generated with 8 parameters are selected, and the total number of samples is 88,000. To further analyze the effectiveness of the proposed model for detecting malicious domain names based on multiple elements, we generate multielement dynamic domain names based on several DGAs and HMM-based SDGAs using elements such as characters, numbers, special characters, two-character combinations, three-character combinations, and words. The domain names generated based on multielement and DGA are named ME-DGA, and the domain names based on multielement and SDGA are named ME-SDGA. The datasets used in the experiments are shown in Table 3.

Based on the above data, several experimental datasets are constructed for testing the detection capability of the proposed algorithm for different types of malicious domains. The detection is binary classification; i.e., a domain name is classified as legitimate or malicious. To evaluate the detection effectiveness of the models, Precision, Recall, Accuracy, and F1-score can be used as evaluation metrics. To calculate these evaluation metrics, the numbers of True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) samples need to be calculated. The metrics are defined as follows:

(i) TP: number of correctly classified malicious dynamic domain names.
TN: number of correctly classified legitimate domain names
FP: number of wrongly classified legitimate domain names
FN: number of wrongly classified malicious dynamic domain names

Using these four terms, the Precision, Recall, Accuracy, and $F_1$-score can be defined and calculated as follows:

(i) **Precision**: The proportion of correctly classified malicious domain samples to all malicious domain samples. It is calculated as

$$\text{Precision} = \frac{TP}{TP + FP} \quad (29)$$

(ii) **Recall**: The proportion of true malicious samples among all samples classified as malicious. It is calculated as

$$\text{Recall} = \frac{TP}{TP + FN} \quad (30)$$

(iii) **Accuracy**: The proportion of correctly classified samples among all samples. It is calculated as

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FN + FP} \quad (31)$$

(iv) **$F_1$-score**: the harmonic mean of Precision and Recall. It is calculated as

$$F_1 - \text{Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (32)$$

Keras is used as the model training framework in this experiment. The machine used in the experiments has a dual-way Intel Xeon E5-2630 V4 CPU, 64 GB of memory, 4T hard disk, and Ubuntu 16.04 operating system.

5.2. **Ablation Experiments**. To verify the effectiveness of the adaptive embedding module, the feature selection module, and the improved dynamic Focalloss proposed in this paper, an ablation experiment containing the comparison of models with different modules is designed. The baseline model in the experiment uses character embedding for the input domain name and then uses four parallel
convolutional branches to extract features, and the results of the four branches are combined and input into a three-layer fully connected layer, and the model is trained using a cross-entropy loss function. The length of the character vector in the model is 128, the number of filters is 128, and the entropy loss function. The average detection accuracy of model 6 represents that the model contains the module.

The comparison detection results of model 1 show that the average detection accuracy of the model improves from 0.9685 to 0.9732 and the F1-score also improves from 0.9684 to 0.9730 after using the adaptive element embedding module; it denotes that the adaptive speed quantization module can significantly improve the detection effect of the model. The average accuracy of model 2 is improved to 0.9724 and the F1-score is also improved to 0.9720, which indicates that the feature selection module can improve the detection effect of the parallel convolutional model. The detection results of model 3 denote that the Focal loss function can improve the model detection effect compared with the cross-entropy loss function. The average detection accuracy of model 4 is 0.9721 and the F1-score is 0.9720, indicating that the improved dynamic Focal loss function can further improve the detection effect of the model. Model 5 achieves an average detection accuracy of 0.9758 and an F1-score of 0.9755, indicating that the use of both the adaptive element embedding module and the feature selection module can effectively improve the detection effect. The average detection accuracy of model 6 can reach 0.9812 and the F1-score can reach 0.9810. It denotes that, after using the adaptive element embedding, feature selection module, and the improved dynamic Focal loss function, the detection accuracy of the model is significantly improved compared with the baseline model.

The ablation experimental results denote that the adaptive domain name element embedding module, feature selection module, and improved dynamic Focal loss can effectively improve the detection effect of the model.

5.3. Comparison Experiments on DGA Dataset. In this subsection, the proposed detection model is compared with five previous detection algorithms, namely, LSTM model [23], Invincea CNN model [25], LSTM-MI model [26], N-gram CNN-based model [29], and HDNN model [28]. The proposed model uses the model 6 structure from the previous subsection which is named MEAE-CNN. The contrasted dataset consists of 20 selected DGA samples and legitimate domain names. The comparison detection results are shown in Table 5.

Among the 20 selected DGA domain name samples, most of them are generated by selecting characters according to a certain random number of seeds, and some of them are generated with less random characters, which are more difficult to detect; for example, Simda and Virut algorithms strictly limit the length of generated domain names and generate domain names with shorter lengths and less randomness. Although Pykpoc and Symmi generate longer domain names, the vowel letters and consonant letters are selected according to their cooccurrence relationship; the generated domain names are more difficult to detect. Matsnu and Suppobox use words to generate domain names with the lowest randomness. Among the compared algorithms, Invincea CNN simply uses a parallel convolutional structure, which cannot effectively extract the association features between characters and therefore has the worst detection effect on DGA domain names, with an average accuracy of 0.9685. In contrast, the detection accuracy of the LSTM model is significantly higher, at 0.9712. After adding a sample equalization strategy to the LSTM model, the accuracy of the LSTM-MI model is slightly improved, and the accuracy is increased to 0.9721, and the recall is improved to 0.9709. N-gram CNN can achieve a more accurate quantitative representation of DGA domain names due to the embedding method of N-gram characters; the accuracy is
higher compared to LSTM-MI. The average accuracy of the HDNN model is further improved due to the use of modules such as parallel convolution, bidirectional LSTM, and an attention mechanism, as well as the use of Focalloss function in the training process, so the accuracy of the detection of DGA samples is further improved. The accuracy of the proposed model is 0.9812 and the recall is 0.982 because of the use of a more adaptive domain embedding module, the use of feature selection module to select features from parallel convolution, and the use of improved dynamic Focalloss function in the training process; it denotes that the proposed modules can significantly improve the detection accuracy. The experimental results denote that the proposed algorithm can significantly improve the detection of DGA domain names.

To further compare the detection effect of these models, the classification accuracy of the models for each domain type is shown in Figure 10. It can be seen that all the comparison algorithms maintain high classification accuracy for most DGA types. For Symmi and Virut, the detection accuracy of N-gram CNN model and HDNN model is significantly higher than that of LSTM and Invincea CNN algorithms, while the proposed algorithm performs the best detection accuracy. For Pykspa and Simda, the detection accuracy of the proposed algorithm is also significantly better than other comparison algorithms. For word-based Matsnu and Suppobox, the detection accuracies of LSTM and Invincea CNN models are both below 0.5, the detection accuracies of LSTM-MI are slightly improved, up to about 0.5, and the detection accuracies of N-gram CNN and HDNN are higher, both over 0.6. The detection accuracy of the proposed algorithms for these two DGA domain names is significantly higher than other comparison algorithms, and the detection accuracy for Suppobox and Matsnu domain names is 0.8 and 0.85, respectively. The mentioned DGA types are difficult to detect and belong to the hard-to-detect samples. Among the comparison algorithms, LSTM-MI improves the loss function considering the sample unbalance and increases the weight of difficult samples. The HDNN absorbs the advantages of LSTM and CNN and extracts domain name features from multiple dimensions; it effectively improves the detection accuracy of several hard-to-detect samples. The proposed algorithm improves domain representation by adaptively segmenting and embedding different elements in domain names, which can improve the accuracy of vector representation of two-character combinations, three-character combinations, and words. When it uses a feature selection module to adaptively select features extracted from multiple convolutional kernel dimensions of domain names and finally uses an improved loss function that can dynamically adjust the loss weights of the hard-to-detect samples. The comparison experiments denote that the detection accuracy of the proposed algorithm is significantly higher than the comparison algorithms on the DGA dataset.

The comparison experiments on the DGA dataset denote that the detection accuracy of the proposed algorithm is significantly higher than five comparison algorithms; in particular, the detection accuracies of some hard-to-detect domain types are improved significantly.

| Dataset type | DGA description | Quantity |
|--------------|----------------|----------|
| DGA          | 20 types of DGAs are collected, namely: Gameover, Dircrypt, Tinba, Necurs, Ramdo, Cryptolocker, Emotet, Corebot, Banjori, Qakbot, Rovnix, Kraken, Ramnit, Locky, Pykspa, Simda, Symmi, Virut, Matsnu, Suppobox | 200,000 |
| SDGA         | 8 types of SDGAs are collected, namely: DNL1, DNL2, DNL3, DNL4, 9ML1, 500KL1, 500KL2, 500KL3 | 88,000 |
| ME-DGA       | ME-DGA: generating domain names based on characters, numbers, special characters, two-character combinations, three-character combinations, and words, using a variety of DGAs | 60000 |
| ME-SDG       | ME-SDG: generating domain names based on characters, numbers, special characters, two-character combinations, three-character combinations, and words, using a variety of SDGAs | |
| Legitimate   | Top 1 million domain names updated daily by Alexa from January 2009 to February 2019 | Prepared in 4:1 quantity with dynamic domain names |

| Model | Adaptive embedding module | Feature selection module | Focalloss | Improve dynamic Focalloss | F1-score | Accuracy |
|-------|----------------------------|--------------------------|-----------|--------------------------|----------|----------|
| Baseline | x                         | x                        | x         | x                        | 0.9684   | 0.9685   |
| Model 1  | o                         | x                        | x         | x                        | 0.9730   | 0.9732   |
| Model 2  | x                         | o                        | x         | x                        | 0.9720   | 0.9724   |
| Model 3  | x                         | x                        | o         | x                        | 0.9710   | 0.9712   |
| Model 4  | x                         | x                        | x         | o                        | 0.9720   | 0.9721   |
| Model 5  | o                         | o                        | x         | x                        | 0.9755   | 0.9758   |
| Model 6  | o                         | o                        | o         | o                        | 0.9810   | 0.9812   |
5.4. Comparison Experiments on SDGA Dataset. Since SDGA uses the HMM model to generate dynamic domain names, the randomness of character combinations is lower and more difficult to detect. To verify the detection accuracy of the proposed algorithm for SDGA samples, we construct a dataset using 8 SDGA domain names with different parameters and a large number of legitimate domain names, and a comparison experiment is conducted with 5 algorithms. The detection results are shown in Table 6.

Among the 5 comparison algorithms, the Invincea CNN model has the highest precision, while its average accuracy and F1-score perform worst. The detection result of the LSTM model is slightly higher than that of the Invincea CNN model, with an accuracy of 0.8481 and an F1-score of 0.8275. After using the sample balance module, the detection accuracy of the LSTM-MI model is improved to 0.8621 and the F1-score is 0.8485. The N-gram CNN model is less effective in detecting SDGA domain names, with an accuracy of only 0.8476. HDNN uses many types of deep learning architectures and is adaptive for SDGA domain names with different parameters; the detection accuracy is significantly improved to 0.8833. The detection effect of the proposed algorithm is significantly better than the comparison algorithms, with an accuracy up to 0.9012 and an F1-score up to 0.8940. The recall is significantly increased to 0.8610 because the model uses the improved dynamic loss function to increase the focus on hard-to-detect samples and its balance strategy leads to a slight decrease in the precision to 0.9296. The comparison results denote that, based on the adaptive embedding module, the feature selection module, and the improved dynamic Focal loss function, the proposed algorithm can effectively improve the detection result on the SDGA dataset.

To further compare the detection effect on SDGA domain names, the classification accuracy of algorithms for each domain type in the SDGA dataset is as shown in Table 7. DNL1~DNL4 are samples generated based on English words. The other SDGA samples are generated based on legitimate domain names. The accuracy of the LSTM for DNL1~DNL4 is around 0.5, and the detection accuracy of the LSTM algorithm for 500KL1 and 500KL2 is 0.9785 and 0.9565, respectively. The accuracy of 9ML1 is only 0.9115, while 500KL3 training samples rely on more forward characters and are therefore more difficult to detect, with an accuracy of only 0.8249. The detection accuracies of Invincea CNN for SDGA domain names are close to the results of LSTM. The LSTM-MI uses a balance module, and the detection accuracy for hard-to-detect samples is higher, so the detection accuracy for DNL1~DNL4 is significantly higher than that of the LSTM model, with an accuracy of 0.6247 for DNL1 and about 0.6 for DNL2~DNL4. The detection accuracy for 500KL1~500KL3 is also higher than the LSTM model, and its shortcoming is that the detection accuracy for legitimate domain names decreases by about 3%. N-gram CNN has detection accuracies for some SDGA types because it uses N-gram embedding; however, the detection accuracy of DNL2~DNL4 and 500KL2~500KL3 increases, while the detection accuracy of DNL1 and 500KL1 decreases.
denotes that the embedding of fixed-length character combinations cannot be adapted to generative models with different parameters. HDNN model significantly improves the detection accuracy of multiple SDGA domain types due to the use of multiple architectures. The detection effectiveness of the proposed algorithm is improved compared to the HDNN algorithm, and the accuracy of each SDGA domain is significantly improved, and the detection accuracy of legitimate domain names is also higher than that of the HDNN model. The experimental results denote that the proposed algorithm can effectively improve the detection accuracy of SDGA domain names, especially the detection accuracy of hard-to-detect samples.

5.5. Comparison Experiments on ME Dataset. The proposed adaptive embedding module can embed many different elements of domain names. Although the DGA and SDGA domain name samples contain characters, words, and character combinations, the multiple elements are not sufficiently mixed. To further compare the detection accuracy of the proposed model for dynamic domain names generated based on multiple elements, we use multiple DGA and SDGA algorithms to generate domain names based on multiple elements. Among the ME-DGA samples, ME-Suppobox is a set of samples generated using the Suppobox algorithm with multiple elements, ME-Kraken is a set of samples generated using the Kraken algorithm with multiple elements, and ME-Corebot is a set of samples generated using the Corebot algorithm with multiple elements. Based on different parameters L in SDGA and multiple elements, ME-DNL1, ME-DNL2, and ME-DNL3 can be designed.

The detection results of the proposed algorithm and the five comparison algorithms for ME-DGA samples are shown in Table 8. The difficulty of detecting dynamic domain names based on multiple elements is enhanced, and the recall of the LSTM algorithm is only 0.6856, with an average accuracy of only 0.8012. The recall and accuracy of the Invincea CNN algorithm are lower than those of the LSTM model, with a recall of only 0.6793 and an accuracy of 0.7980. The LSTM-MI model has better detection results than the LSTM model, with a recall of 0.7350 and average accuracy of 0.8310. N-gram CNN has slightly better detection results than Invincea CNN due to the use of adjacent N-gram character embedding, with an accuracy of 0.8043 and recall of 0.7612. The HDNN model benefits from combining the advantages of LSTM and CNN structures, and its detection accuracy is higher than that of the LSTM-MI model, with an average accuracy of 0.8421 and recall of 0.7612. The proposed MEAE-CNN model performs significantly better than other models due to the multiple elements embedding and the use of multiple optimization modules. The average accuracy can reach 0.8920 and the recall can reach 0.8407. The comparison results denote that the proposed adaptive embedding module, the feature selection module, and the improved dynamic Focalloss function can effectively improve the detection result on the multielement malicious domains.

To further compare the detection effectiveness of the algorithms, the detection accuracy for each type of dynamic domain name is shown in Table 9. It can be seen that the detection accuracy for ME domain names is lower than that for DGA domain names and SDGA domain names, indicating that the detection difficulty of ME domain names is higher than that of DGA and SDGA domain names. Among the detection models, the detection accuracy of the LSTM model for ME-DNL1–ME-DNL3 is only about 0.50 and about 0.87 for the three ME-DGA domain names. The detection accuracy of the Invincea CNN model for most ME-DGA and ME-SDGA domain names is lower than that of the LSTM model. The detection accuracy of the LSTM-MI model is significantly higher than that of the LSTM model for all three ME-SDGA and three ME-DGA domain names. The N-gram CNN model improves the detection accuracy of ME-DNL2 and slightly improves the detection accuracy of three ME-DGA domain names compared with the Invincea CNN model, indicating that the N-gram embedding can improve the domain representation of some

### Table 5: Comparison results on DGA dataset.

| Detection algorithm | Precision | Recall | F1-score | Accuracy |
|---------------------|-----------|--------|----------|----------|
| LSTM                | 0.9730    | 0.9690 | 0.9710   | 0.9712   |
| Invincea CNN       | 0.9706    | 0.9662 | 0.9684   | 0.9685   |
| LSTM-MI             | 0.9731    | 0.9709 | 0.9720   | 0.9721   |
| N-gram CNN          | 0.9724    | 0.9746 | 0.9735   | 0.9736   |
| HDNN                | 0.9730    | 0.9770 | 0.9750   | 0.9762   |
| MEAE-CNN (proposed) | 0.9799    | 0.9821 | 0.9810   | 0.9812   |

### Table 6: Comparison results on SDGA dataset.

| Detection algorithm | Precision | Recall | F1-score | Accuracy |
|---------------------|-----------|--------|----------|----------|
| LSTM                | 0.9575    | 0.7286 | 0.8275   | 0.8481   |
| Invincea CNN       | 0.9686    | 0.7144 | 0.8233   | 0.8456   |
| LSTM-MI             | 0.9416    | 0.7721 | 0.8485   | 0.8621   |
| N-gram CNN          | 0.9640    | 0.6993 | 0.8106   | 0.8476   |
| HDNN                | 0.9176    | 0.8421 | 0.8782   | 0.8833   |
| MEAE-CNN (proposed) | 0.9296    | 0.8610 | 0.8940   | 0.9012   |
multielement dynamic domain names. The HDNN model benefits from the hybrid architecture of CNN and LSTM; the detection accuracy of each type of ME-DGA and ME-SDGA domain name is higher than that of the LSTM-MI model. The detection accuracy of the proposed algorithm for each dynamic domain name is significantly higher than other algorithms, indicating that the proposed adaptive element embedding module can effectively improve the detection accuracy of multielement dynamic domain names.

The proposed algorithm is compared with five algorithms on the ME-DGA and ME-SDGA datasets, and the experimental results denote that the detection accuracy of the proposed algorithms for multielement dynamic domain names is higher than that of the comparison algorithms, and the detection accuracy of each domain type is also significantly improved.

In this section, we constructed 3 datasets and compared the proposed algorithm with five algorithms in several dimensions. The experimental results denote that the detection accuracy of the proposed algorithm on the DGA dataset, SDGA dataset, and ME dataset is higher than the comparison algorithms, indicating that the proposed adaptive embedding module, feature selection module, and improved dynamic Focalloss can effectively improve the detection effect on dynamic domain names.

6. Conclusion
In this paper, a domain name syntax model is proposed from the perspective of element composition and syntax analysis of domain names, and a detection model based on element adaptive embedding is proposed. The detection model uses an adaptive embedding module to segment the domain name into elements and embedding different types of elements, then feeds the embedding results into a parallel convolutional model, and uses a feature selection module to select the convolutional features obtained from different convolutional kernels. To improve the detection ability of the model for hard-to-detect samples, we propose an improved dynamic Focalloss that can dynamically adjust the loss of hard-to-detect samples during the training phase, which can improve the model training effect. A variety of experiments is designed based on public datasets and the proposed algorithm is compared with five algorithms. The experimental results denote that the detection accuracy of the proposed algorithm on 3 datasets is higher than that of the comparison algorithms; in particular, the detection accuracy of hard-to-detect samples is significantly improved. The detection object of this paper does not consider the dynamic domain names generated based on GAN; we will research the adversarial detection of dynamic domain names generated by GAN in the future.
Data Availability
https://data.mendeley.com/datasets/y8ph45msv8/1.

Conflicts of Interest
The authors declare that they have no conflicts of interest.

Acknowledgments
This work was supported in part by the National Natural Science Foundation of China under Grant (U1836104, 61702235) and the Fundamental Research Funds for the Central Universities under Grant 30918012204.

References
[1] L. Yang, P. Li, X. Yang, and Y. Y. Tang, “A risk management approach to defending against the advanced persistent threat,” IEEE Transactions on Dependable and Secure Computing, vol. 17, no. 6, pp. 1163–1172, 2018.
[2] M. Singh, M. Singh, and S. Kaur, “Issues and challenges in DNS-based botnet detection: a survey,” Computers & Security, vol. 86, pp. 28–52, 2019.
[3] A. K. Sood and S. Zeadally, “A taxonomy of domain-generation algorithms,” IEEE Security & Privacy, vol. 14, no. 4, pp. 46–53, 2016.
[4] S., https://blog.checkpoint.com/wp-content/uploads/2015/07/Matsnu-malwareid-technical-brief.pdf Matsnu: Malware ID. check point blog post 2015, 2015.
[5] J. Geffner, “End-to-end analysis of a domain generating algorithm malware family,” Black Hat USA, vol. 2013, 2013.
[6] H. Crawford and J. Aycock, “Kwyjibio: automatic domain name generation,” Software: Practice and Experience, vol. 38, no. 14, pp. 1561–1567, 2008.
[7] Y. Fu, L. Yu, O. Hambolu et al., “Stealthy domain generation algorithms,” IEEE Transactions on Information Forensics and Security, vol. 12, no. 6, pp. 1430–1443, 2017.
[8] X. Yun, J. Huang, Y. Wang, T. Zang, Y. Zhou, and Y. Zhang, “Khaos: an adversarial neural network DGA with high anti-detection ability,” IEEE Transactions on Information Forensics and Security, vol. 15, pp. 2225–2240, 2019.
[9] S. Yadav, A. K. Reddy, A. N. Reddy, and S. Ranjan, “Detecting algorithmically generated malicious domain names,” in Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, pp. 48–61, Melbourne, Australia, November 2010.
[10] S. Yadav, A. K. Reddy, A. L. N. Reddy, and S. Ranjan, “Detecting algorithmically generated domain-flux attacks with DNS traffic analysis,” IEEE/ACM Transactions on Networking, vol. 20, no. 5, pp. 1663–1677, 2012.
[11] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, “Phoenix: DGA-based botnet tracking and intelligence,” in Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 192–211, Springer, Egham, UK, July 2014.
[12] J. Raghuram, D. J. Miller, and G. Kesidis, “Unsupervised, low latency anomaly detection of algorithmically generated domain names by generative probabilistic modeling,” Journal of Advanced Research, vol. 5, no. 4, pp. 423–433, 2014.
[13] D.-T. Truong and G. Cheng, “Detecting domain-flux botnet based on DNS traffic features in managed network,” Security and Communication Networks, vol. 9, no. 14, pp. 2338–2347, 2016.
[14] D. L. Schales, J. Jang, T. Wang et al., “Scalable analytics to detect DNS misuse for establishing stealthy communication channels,” IBM Journal of Research and Development, vol. 60, no. 4, 2016.
[15] L. Yang, J. Zhai, W. Liu et al., “Detecting word-based algorithmically generated domains using semantic analysis,” Symmetry, vol. 11, no. 2, p. 176, 2019.
[16] M. Zago, M. Gil Pérez, and G. Martínez Pérez, “Scalable detection of botnets based on DGA,” Soft Computing, vol. 24, no. 8, pp. 5517–5537, 2020.
[17] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, “Exposure: a passive dns analysis service to detect and report malicious domains,” ACM Transactions on Information and System Security, vol. 16, no. 4, pp. 1–28, 2014.
[18] Y. Shi, G. Chen, and J. Li, “Malicious domain name detection based on extreme machine learning,” Neural Processing Letters, vol. 48, no. 3, pp. 1347–1357, 2018.
[19] M. Mowbray and J. Hagen, “Finding domain-generation algorithms by looking at length distribution,” in Proceedings of the 2014 IEEE International Symposium on Software Reliability Engineering Workshops, pp. 395–400, IEEE, Naples, Italy, November 2014.
[20] K. Kim, J. Lee, H. Lee, and A. Perrig, “PsyBoG: a scalable botnet detection method for large-scale DNS traffic,” Computer Networks, vol. 97, pp. 48–73, 2016.
[21] S. Yadav and A. N. Reddy, “Winning with DNS failures: strategies for faster botnet detection,” in Proceedings of the International Conference on Security and Privacy in Communication Systems, pp. 446–459, Springer, London, UK, September 2011.
[22] M. Antonakakis, R. Perdisci, Y. Nadji et al., “From throwaway traffic to bots: detecting the rise of DGA-based malware,” in Proceedings of the Presented as Part of the 21st [USENIX] Security Symposium (USENIX) Security 12), pp. 491–506, Bellevue, WA, USA, August 2012.
[23] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, “Predicting domain generation algorithms with long short-term memory networks,” 2016, https://arxiv.org/abs/1611.00791.
[24] B. Yu, D. L. Gray, J. Pan, M. De Cock, and A. C. Nascimento, “Inline DGA detection with deep networks,” in Proceedings of the 2017 IEEE International Conference on Data Mining Workshops (ICDMW), pp. 683–692, IEEE, New Orleans, LA, USA, November 2017.
[25] B. Yu, J. Pan, J. Hu, A. Nascimento, and M. De Cock, “Character level based detection of DGA domain names,” in Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8, IEEE, Rio de Janeiro, Brazil, July 2018.
[26] D. Tran, H. Mac, V. Tong, H. A. Tran, and L. G. Nguyen, “A LSTM based framework for handling multiclass imbalance in DGA botnet detection,” Neurocomputing, vol. 275, pp. 2401–2413, 2018.
[27] Y. Qiao, B. Zhang, W. Zhang, A. K. Sangaih, and H. Wu, “DGA domain name classification method based on long short-term memory with attention mechanism,” Applied Sciences, vol. 9, no. 20, pp. 4205, 2019.
[28] L. Yang, G. Liu, Y. Dai, J. Wang, and J. Zhai, “Detecting stealthy domain generation algorithms using heterogeneous deep neural network framework,” IEEE Access, vol. 8, no. 99, pp. 82876–82889, 2020.
[29] C. Xu, J. Shen, and X. Du, “Detection method of domain names generated by DGAs based on semantic representation
and deep neural network,” *Computers & Security*, vol. 85, pp. 77–88, 2019.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, pp. 1–12, Scottsdale, AZ, USA, May 2013.

[31] X. Li, W. Wang, X. Hu, and J. Yang, “Selective kernel networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 510–519, Long Beach, CA, USA, June 2019.

[32] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2018.

[33] T. Zhou and J. Bilmes, “Minimax curriculum learning: machine teaching with desirable difficulties and scheduled diversity,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, pp. 1–23, Vancouver, Canada, April 2018.

[34] M. Zago, M. Gil Perez, and G. Martinez Perez, “UMUDGA: a dataset for profiling DGA-based botnet,” *Computers & Security*, vol. 92, p. 101719, 2020.