LM-cAPI:A Lite Model Based on API Core Semantic Information for Malware Classification

Yifan Zhou, Zhenyan Liu(✉), Jingfeng Xue, Yong Wang, and Ji Zhang

School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China
zhenyanliu@bit.edu.cn

Abstract. Currently, malware is continually evolving and growing in complexity, posing a significant threat to network security. With the constant emergence of new types and quantities of malware coupled with the continuous updating of dissemination methods, the rapid and accurate identification of malware as well as providing precise support for corresponding warning and defense measures have become a crucial challenge in maintaining network security. This article focuses on API call sequences in malware that can characterize the behavioral characteristics of malware as text and then uses the latest text classification-related technologies to achieve the classification of malware. This article proposes a flexible and lightweight malicious code classification model based on API core semantic information. To address the issues of prolonged training time and low accuracy caused by excessive noise and redundant data in API call sequences, this model adopts an intimacy analysis method based on a self-attention mechanism for key information extraction. To enhance the capture of semantic information within malware API call sequences, a feature extraction model based on a self-attention mechanism is used to transform unstructured key API sequences into vector representations, extract core features, and finally connect to the TextCNN model for multi classification. In the dataset of the "Alibaba Cloud Security Malicious Program Detection" competition, the F1 value reached 90% in eight category classification tasks. The experimental results show that the model proposed in this article can achieve better results in malware detection and multi-classification.

Keywords: Network Security · Malware Classification · API call sequence

1 Introduction

Malware is one of the most serious threats to network security, serving as a key attack carrier in various network security events. When malware is executed, it

Supported by Major Scientific and Technological Innovation Projects of Shandong Province (2020CXGC010116) and the National Natural Science Foundation of China (No. 62172042).

M. Andreoni (Ed.): ACNS 2024 Workshops, LNCS 14586, pp. 29–42, 2024.
https://doi.org/10.1007/978-3-031-61486-6_3

poses a risk to the confidentiality, availability, and integrity of sensitive information and data in the target system. Moreover, in order to avoid traditional malware detection and eradication mechanisms (such as firewalls, antivirus software, and other signature-based defense methods) and improve their own survivability, malware programmers employ sophisticated techniques. These involve modifying and confounding malicious samples within the same family using diverse strategies to alter code structures and generate various different code variants while maintaining semantic equivalence. There are certain similarities in structure and behavior among variant samples from the same family. As malware spreads, it utilizes various deformation engines to automatically generate new variants. Simultaneously, the development of the malware industry chain is also continuously collectivized and organized. Overall, the above phenomenon has resulted in the proliferation of malware not only in terms of quantity but also in the diversification of defense evasion methods. Consequently, the need for automated detection, elimination, and tracing of malware has become increasingly urgent.

Over the past few years, the volume of malware data has grown rapidly. Within the realm of artificial intelligence, natural language processing (NLP) has emerged as a mature subfield, with machine learning [1] and neural network methods of natural language processing gradually reaching maturity in the domain of malware detection. Machine-learning-based malware detection methods [2,3] can automatically analyze a large amount of data through inductive reasoning, enabling the detection and classification of malware into families. The essence of machine-learning-based methods lies in feature extraction and model building. The feature extraction process can be achieved through both static and dynamic analysis methods. Commonly used features include opcode sequences, API call sequences, byte sequences, etc. [4–6]. The model classifies samples by analyzing features and using algorithms such as classification or clustering. An API serves as the interface between an application program and a system. Its call sequence encapsulates the behavioral information of the code during actual runtime, providing an accurate characterization of the program's purpose. Through the analysis of API call sequences, it becomes evident that malware typically calls fixed API sequences to perform destructive behavior. With the continuous development of technology in the field of natural language processing, API sequences can be regarded as a form of semantic text, exhibiting temporal relationships between APIs. The relevant technologies of natural language processing can be applied to analyze API sequences [5,7].

We present a versatile and lite malware classification model based on the key semantic information of API call sequences. To address the issue of long training time and low accuracy resulting from excessive noise and redundant data in API call sequences, this model employs keyword extraction technology for key information extraction. To enhance the extraction of semantic information from API call sequences, this article extensively employs language-training language models to obtain rich semantic representations. Moreover, neural network models are employed to address the multi-classification challenge posed by malware. In this

experiment, we used the dataset provided by the "Alibaba Cloud Tianchi Competition Security malware Detection;; competition question and conducted the necessary data processing. The experimental findings demonstrate the superiority of our proposed method in comparison to the general classification methods using API call sequences as features. The proposed method is more effective in multi classification of malware, with an accuracy rate of 90%.

2 Related Work

Malware technology has caused significant harm to users, enterprises, and even countries due to its continuous development. Numerous information security researchers both domestically and internationally are dedicated to the research of malware.

The core of machine-learning-based malware detection methods lies in feature extraction and modeling. The model classifies samples by analyzing features and using algorithms such as classification or clustering. An API serves as the crucial interface between an application program and a system, and its call sequence can substantially reflect the program's behavior. Therefore, numerous malware analyses are based on API sequences. Darshan [8] et al. extracted API call sequences from JSON files obtained from sandbox operations. They then applied the N-Gram method to process the sequences and used machine learning algorithms to construct classifiers with high detection accuracy. However, a drawback of this method is its consideration of only a small subset of features from a larger pool, necessitating further improvement in accuracy. Fang Yong [9] et al. addressed this limitation by mixing dynamic and static API features through weight ratios to compensate for the shortcomings of a single feature. They also proposed a new semi-supervised clustering algorithm based on the unsupervised DBSCAN algorithm, significantly improving the accuracy of clustering.

The above methods all require a large amount of data and labor to ensure accurate classification. Furthermore, some models require the use of manual design feature extraction, resulting in serious limitations in generalization issues. The use of neural networks based on deep learning methods to solve text classification problems is currently a hot research topic. Deep learning [10] is a branch of machine learning based on multi-layer neural networks to learn deeper features in samples. It is a complex machine learning algorithm capable of automatically extracting the features of malware through multi-layer neural networks, simplifying the feature extraction process and enhancing detection accuracy. Lu Xiaofeng [11] et al. proposed a model assembly method. They introduced a correlation analysis algorithm for API calls to mine features of API sequences. Machine learning algorithms were then employed to learn these features. Subsequently, a recurrent neural network was utilized to detect malware, and finally, a model combination was conducted, resulting in improved outcomes. However, its drawback is that recurrent neural networks are unstable when dealing with long sequences, potentially leading to extended model training time and poor detection performance. Cui [12] et al. used grayscale images to represent disassembly files of malware. They leveraged the advantages of convolutional neural

networks in image processing to recognize and classify grayscale images and used bat algorithms to address the problem of data imbalance between different malicious software families. Nevertheless, a drawback is that the model exhibits low flexibility and requires setting the input images of all samples to a uniform size.

The malware classification method based on deep learning does not require the use of manually designed feature extraction and has high classification accuracy. However, the training time of neural networks is long, and they may generate a large number of parameters, resulting in excessive hardware costs. Currently, it is impossible to avoid the problem of using deep learning for prediction. To address the issue of a large number of malware variants while also considering detection efficiency and effectiveness, this article combines the characteristics of the target task and the data used, takes API call sequences as the research object, and regards them as a piece of text with semantic information. Additionally, it implements a lightweight model based on API core semantic information using the self-attention mechanism, which enhances the accuracy of multi-classification.

3 A Lite Model Based on API Core Semantic Information

Figure 1 illustrates process of malware classification based on deep learning using API call sequences as the research object. Firstly, collect executable programs on the Windows platform, encompassing both malicious and benign samples. Subsequently, utilize the Cuckoo sandbox environment for simulation execution. After certain data processing, obtain API sequences. Then, classify them using a classification model. Finally, process the text data for classification and input it into the trained model to obtain the malware classification results. The lightweight nature of the model proposed in this article is evident in two aspects. Firstly, the model's input consists of API call sequences containing only key information. Secondly, while ensuring the model's effectiveness, the number of parameters in the model is greatly reduced.

3.1 API Call Sequence

The malware classification task based on API call sequences utilizes API call log files as analysis objects. It extracts features from the collected files using text analysis methods, preprocesses the data, and incorporates other techniques for feature selection. However, there are significant differences between API call log files and real-world text files. The first distinction lies in the fact that data in text files is often in common languages such as Chinese or English, encompassing Chinese and English words, etc. In contrast, data in API call log files represents API functions, serving as the interface between the application program and the Windows system, with a specific nature. Therefore, when classifying malware using a pre-trained model with good training results, such as the powerful BERT, it is necessary to extract API functions from the API sequence and establish a special API vocabulary to retrain the language model. The second difference arises

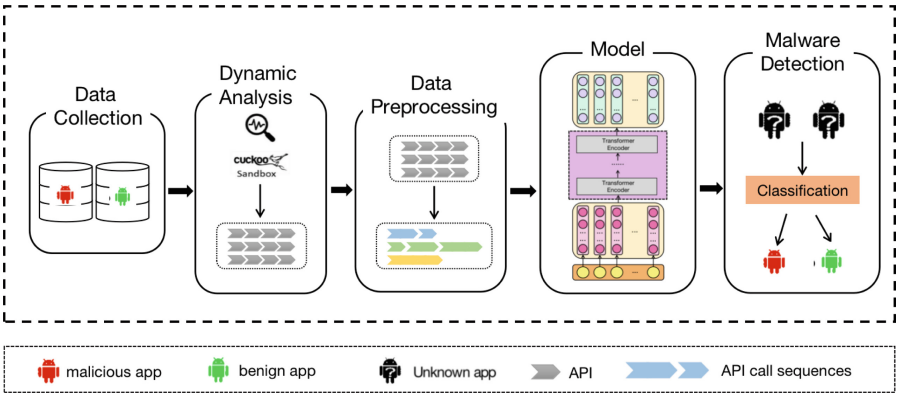

Fig. 1. Malware Classification Process.

from the fact that, to avoid analysis, malware often inserts a significant amount of redundant behavior into normal behavior. This results in excessively long API sequences that interfere with analysis and conceal the malicious intent of the code. Therefore, before extracting the core semantic information of API call sequences, it is necessary to perform data preprocessing operations to remove redundant and noisy data from the original sequence. This article reprocesses API call sequences to reduce their complexity. However, the data preprocessing method only removes multiple duplicate sequences, solving the redundancy problem in file format. At this point, a substantial amount of information in the API call sequence still has little impact on the classification results. Therefore, this article proposes an operational method for extracting core semantic information from the noise information in the API call sequence after deduplication. Considering that the classification process of malware often relies on a few key information points to obtain the classification results, it is necessary to perform key information extraction and feature selection operations on the API call sequence after preprocessing the extensive data and ultimately extracting its core semantic information.

3.2 A Lite Method for Extracting Key Semantic Information Based on BERT

Since API call sequences fall into the category of a special text sequence, they can be considered unstructured data. When classifying malware, it is essential to convert these sequences into vector form. Therefore, a word embedding layer is needed for vector representation.

The advent of the Transformer model has overcome the shortcomings of using convolutional neural networks and recurrent neural networks in malware classification. These traditional models, being sequence-dependent, are constrained to unidirectional semantics and lack the ability to simultaneously utilize contextual

information. The Transformer model addresses these challenges by integrating a self-attention mechanism into the encoder-decoder framework.

This article designs an intimacy analysis method utilizing the self-attention mechanism. The method calculates the intimacy of word vectors and API call sequences with varying lengths, identifying high intimacy sequences as key sequences. The specific method involves inputting the API call sequence into the BERT model for extraction and embedding, resulting in a vector representation of the API call sequence. Subsequently, N-gram is used to extract word vectors of varying lengths, and cosine similarity is used to identify the phrase that is most similar to the original API call sequence. The higher the cosine similarity, the higher the intimacy. Finally, the sequence with the highest affinity for the API call sequence is identified as a key semantic sequence. The core semantic feature information is then extracted from this key semantic sequence as input.

3.3 A Lite Method for Core Semantic Information Based on BERT

API call sequences are considered unstructured data due to the fact that they belong to a distinct category of text sequences. When classifying malware, it is necessary to convert it into vector form, which necessitates the use of a word embedding layer for vector representation. The Transformer model, by bypassing the limitations associated with autoregressive models in feature extraction, has the capability to comprehensively learn any dependency relationships mentioned in the previous text. This article uses a simplified and improved ALBERT model as a feature extractor and then uses a classification model to achieve multi-classification of malware. There are several explanations for the feature extraction model:

Model Input: The model converts each word into a vector as input, establishing a word vector table. The original text is tokenized, and [CLS] is inserted at the beginning to indicate that the feature is used for the classification model. In the model's final layer, the corresponding vector of this bit can serve as the semantic representation of the entire sentence. This is because compared to other words already in the text, this symbol without obvious semantic information will more "fairly" integrate the semantic information of each word in the text. As a result, it adeptly represents the semantics of the entire sentence. The key semantic sequences are input into the Embedding and Encoder layers of the BERT model to obtain an embedded representation containing the core semantic information.

Word Embedding: The vector of word embedding relies on word mapping, and it learns contextually independent representations. The output value from the feature extractor not only encompasses the word's own semantics but also incorporates contextual semantics. It learns contextual representations and should contain more semantic information. Consequently, the BERT model's Encoder

should yield a larger vector dimension to accommodate more semantic information. In this article, the word embedding dimension E (API) of the model is 128 dimensions, while the vector dimension T (API) output by the Encoder encoder of the BERT model is 384 dimensions. E (API) is much smaller than T (API). When processing API call sequences, there are a total of 295 categories of API functions, resulting in a vocabulary size V (API) of 295. The specific operation of the word embedding layer is to input a vector with dimension V (API) into a low-dimensional word embedding matrix, map it to a low-dimensional space with dimension E (API), and then input a low-dimensional word embedding matrix with dimension E (API) into a high-dimensional word embedding matrix, and finally map it to a T (API) dimensional word embedding. Dimensionality reduction operations significantly reduce the number of parameters in the model. The Eqs. (1) and (2) show the change in time complexity after word embedding matrix decomposition.

$$O = V(API) * T(API) \tag{1}$$

$$O = V(API) * E(API) + E(API) * T(API) \tag{2}$$

Layer Parameters: In the BERT model, the sharing of parameters is limited to either the fully connected layer or the attention layer. This article incorporates parameter sharing between several layers to further minimize training parameters and enhance training time. Specifically, the multi-head attention layer and the fully connected feedforward neural network layer share parameters. The parameter size of the feature extractor can be greatly reduced by using an improved self-attention mechanism-based core semantic extraction method, the overall computational speed of the model can be accelerated, the hardware memory overhead can be reduced, the training speed can be accelerated, and the risk of model degradation can be reduced. The feature extractor presented in this paper is more flexible and lightweight when compared to the steps of extracting features in a large pre-trained language model.

3.4 TextCNN Classification Model

Extract the features of the sentence by inputting the embedded representation containing sufficient semantic information into the convolutional layer of the TextCNN model.

The feature maps are then input into the TextCNN model's maximum pooling layer, where they are concatenated to form a vector representation, resulting in a one-dimensional vector. Subsequently, the ReLU activation function is used to output, and a dropout layer is added to prevent overfitting. The fully connected layer is responsible for establishing the relationship between feature information and category information. Finally, all fully connected layer output values are connected to the softmax layer, and multi-classification results are output.

Finally, the overall structure of the model is shown in Fig. 2, which is mainly composed of an input layer, an extraction core sequence layer, a feature extraction layer, a TextCNN layer, and an output layer. Initially, the API call sequence is input from the input layer to the core sequence layer for extracting key information, shortening the data length, and reducing the data volume. The feature extraction layers are then connected to extract sufficient semantic representation while significantly reducing the training time. After encoding, the text of the API call sequence, similar to text, is converted into serialized data, which is then fed into the Transformer encoder. The final feature vector representation of the output text is obtained after training with a self-supervised multi-layer bidirectional Transformer encoder. It then enters TextCNN's convolutional layer to extract the feature representation of the sentence, obtain the feature map, and connect the maximum pooling layer. The one-dimensional vector input is a fully connected layer after the pooling procedure. Finally, all fully connected layer output values are connected to the softmax layer, and multi-classification results are output.

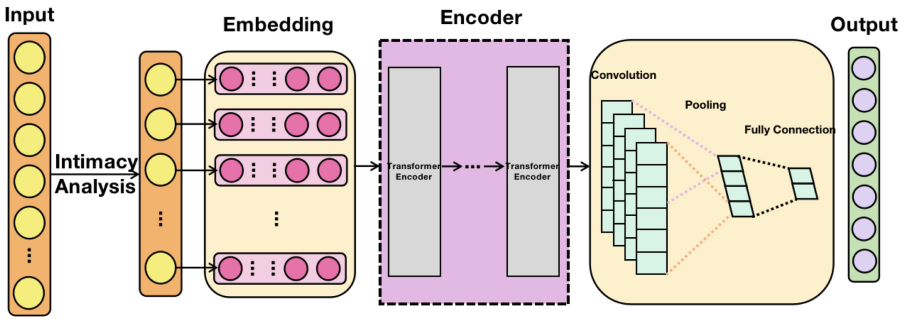

Fig. 2. A Lite Malware Classification Model Based on API Core Semantic Information.

The model incorporates key information extraction mechanisms, self-supervised learning mechanisms, and a simple convolutional neural network to form an overall model. Through the above improvement techniques, the model greatly reduces the large number of parameters generated during the self-supervised learning process and significantly accelerates its training speed. After conducting numerous experiments, it was discovered that the above enhancements effectively improve the accuracy of model predictions. This indicates that relevant improvements are very necessary.

4 Experimental Validation and Result Analysis

4.1 Dataset

The dataset is provided by the Alibaba Cloud Tianchi Competition Security Malicious Program Detection Challenge, which is derived from the API instruc-

tion sequence of a Windows binary executable program simulated by a sandbox program. The sample data provided in the question was obtained from the internet. The types of malicious files include infectious viruses, Trojan programs, mining programs, DDOS Trojan horses, ransomware, etc., totaling 600 million pieces of data. In consideration of the experiment's utilization of call sequences distinguished by file numbers, there are a total of 13887 files. The training set was divided into 11110 and 2777 pieces, with a total of 295 API functions counted, according to the 8:2 ratio. Table 1 illustrates the distribution of sample types and specific quantities.

Table 1. Types and quantity of Malware samples.

Label	Type	Quantity
0	**Benign sample**	4978
1	**Ransomware virus**	502
2	**Mining procedures**	1196
3	**DDoS Trojan**	820
4	**worm-type virus**	100
5	**Infectious virus**	4289
6	**Backdoor program**	515
7	**Trojan program**	1487

4.2 Data Preprocessing

To elude analysis, malware frequently injects a significant amount of redundant behavior into normal operations, resulting in the presence of multiple consecutive identical APIs or API sequence fragments in the sequence. This redundancy in information causes the resulting API sequence to become overly lengthy. This not only hampers analysis and conceals the malicious intent of the code but also extends the training time. Therefore, this article aims to reprocess API call sequences, diminishing their complexity and yielding API call sequences that genuinely reflect program behavior.

Additionally, manage APIs that convey the same meaning but have distinct function names within Windows APIs. For instance, LoadLibraryA and LoadLibraryW are both library loading functions that end with A and W, respectively. The reason is that the system provides different APIs for different encodings, with the W ending mainly for UNICODE encoding and the A ending mainly for ASCII encoding format. For such functions, consider the approach of eliminating the suffix, presenting both LoadLibraryA and LoadLibraryW as LoadLibrary.

We analyzed the distribution of data length before and after data preprocessing. Among these, 4,806 samples exhibit API sequence lengths within 500, while 2,769 samples have API lengths exceeding 10000, accounting for 19.9 of the total samples. There are 6216 API sequences with a length less than 500,

and only 827 with a length greater than 10000, accounting for 5.9 of the total. Through statistical analysis, it is evident that the deduplication operation effectively reduces the length of API sequences, which helps to improve subsequent analysis efficiency.

4.3 Evaluation Indicators

According to the statistics of different categories of malware in the dataset used, it can be seen that the data distribution is uneven. There are over 4000 benign samples in the training set, although the minimum number is less than 100. Therefore, the text adopts accuracy P, recall R, f1 value, and weighted average value as evaluation indicators for the malware classification model, which are all calculated based on TP, TN, FP, and FN. TP denotes predicting positive cases as positive cases; FN denotes predicting positive cases as negative cases; FP denotes predicting negative cases as positive cases; and TN denotes predicting negative cases as negative cases.

Based on the values of the above indicators, calculate the accuracy, recall, and value of the i-th Malware category using formulas (3), (4), and (5). Set the number of data items for the i-th Malware category as, and the total number of data items as N. Then, calculate the weighted average P, weighted average R, and weighted average value using Eq. (6) (7) (8):

$$P_i = \frac{TP_i}{TP_i + FP_i} \tag{3}$$

$$R_i = \frac{TP_i}{TP_i + FN_i} \tag{4}$$

$$F_i = \frac{2 \times P_i \times R_i}{P_i + R_i} \tag{5}$$

$$P = \sum P_i \times \frac{N_i}{N} \tag{6}$$

$$R = \sum R_i \times \frac{N_i}{N} \tag{7}$$

$$F = \sum F_i \times \frac{N_i}{N} \tag{8}$$

4.4 Experimental Results

Parameter Settings. The key to handling sequence problems with TextCNN is to use convolution to express sequence information. It is a one-dimensional convolution, and using a single-length convolution kernel may lose some feature information. Therefore, this article sets up a TextCNN model to extract features from various angles using several convolution kernels of varying sizes, thereby increasing the comprehensiveness of the features. This article conducts experimental comparisons and designs convolutional kernels with sizes of 4, 5, 6, and

7. Each convolutional kernel contains 128 neurons, and ReLU is used as the activation function. A maximum pooling approach is utilized to reduce dimensionality after each convolutional layer. The concatenate function is then applied to combine numerous convolutional and pooling layers. The cross-entropy loss function is utilized to calculate the loss value during model training, and the Adamw optimization method is used to achieve gradient descent and update the model parameters. Setting the batch size to 32 when using the batch training method, which divides the entire dataset into several small datasets, helps the model converge and alleviates the problem of falling into local optima. Set the number of iterations for training, i.e., the epoch value, to 10 and save the optimal model for comparative analysis. Finally, it was determined that the best classification performance was achieved when the improved convolutional neural network parameters were taken from Table 2.

Table 2. LM-cAPI Parameter settings

Label	Type
$embedding_size$	128
$hidden_size$	384
$learning_rate$	5e–5
$filter_sizes$	[4,5,6,7]
$num_filters$	128
$classifier_dropout_prob$	0.1
num_train_epochs	10
$batch_size$	32

Experimental Results and Analysis. According to the comparative experimental results in Table 3, it can be seen that this article established a dictionary based on API call functions and retrained the language model, achieving good multi-classification results. Furthermore, the model introduces and optimizes key information extraction and self-supervised learning methods. From a technical perspective, it is beneficial to learn as much semantic information as possible from the API sequence. This will significantly reduce the problem of parameter explosion caused by self-supervised learning while ensuring the accuracy of model classification.

We draws the loss index and accuracy change curves of the model on the training and validation sets, as shown in Fig. 3. In the training process of malicious code multi classification, when the loss is generally between 0.1 and 0.2, the model has basically converged.

Table 3. Comparison of experimental results

model	precision	recall	f1-score
textcnn	0.80	0.79	0.81
bert	0.70	0.69	0.69
lm-capi	0.93	0.88	0.90

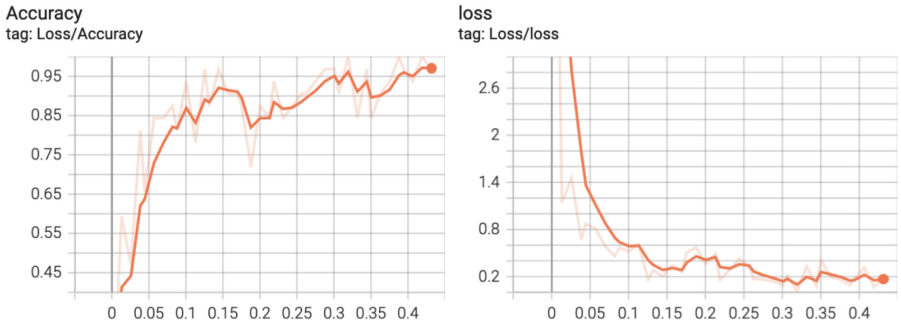

Fig. 3. The loss index and accuracy change curve of the model on the training and validation sets.

Ablation Experiment. This section will split each module and conduct a series of ablation experiments on the AAPD dataset to verify Verify the effectiveness of each module of LSGG. It is mainly divided into the following five parts:

(1) TextCNN: Remove the word embedding part of the LM-cAPI to directly interact with the label text in TextCNN.
(2) B-TCNN: Only BERT and TextCNN are used, excluding the shared parameter mechanism and word embedding layering mechanism of LM-cAPI. In the neural network part, the English BERT pre training model is used.
(3) Pre B TCNN: Only BERT and TextCNN are used, excluding the shared parameter mechanism and word embedding layering mechanism of LM-cAPI. In the neural network part, in order to ensure fairness, the BERT model is retrained for special datasets to obtain a pre trained language model based on API call sequence vocabulary.
(4) M-cAPI: does not perform core semantic extraction, removes the core semantic extraction part of LM-cAPI, and directly classifies based on the cleaned dataset.
(5) LM cAPI: The standard LM cAPI. As shown in Fig. 4, it can be concluded that pre-B-TCNN, M-cAPI, and LM cAPI perform better than B-TCNN in multiple indicators. This implies that when employing pre-trained language models for API call sequences, a new vocabulary needs to be used to re-train the model.

Moreover, the performance of pre-B-TCNN has seen a slight decline when compared to TextCNN. This suggests that the BERT model is limited to handling text data with a length of 512. The truncated text loses some key data,

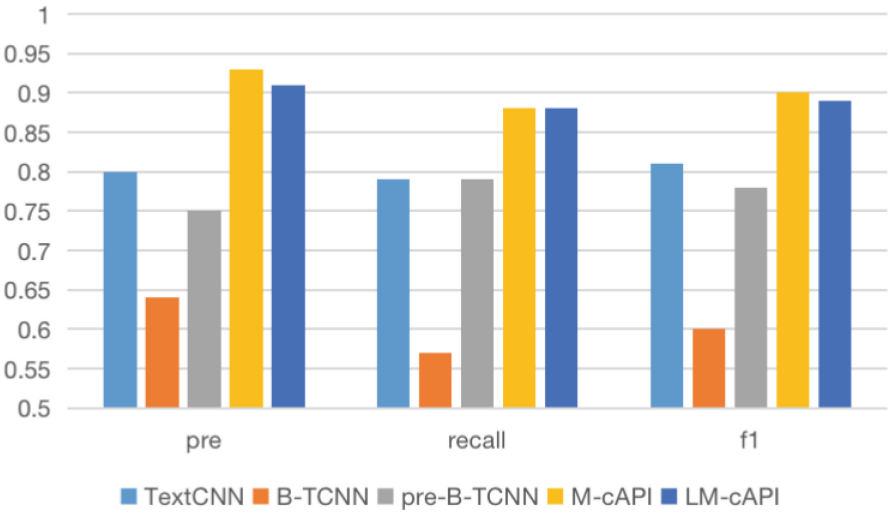

Fig. 4. Schematic diagram of LM-cAPI ablation experiment.

potentially leading to a decrease in results. In addition, the training of pre-trained language models is more time-consuming, necessitating the use of a lightweight model.

5 Conclusion

In the face of the rapid proliferation of malicious code types and quantities, coupled with the continuous updating of dissemination methods, the challenge of promptly and accurately identifying malicious code stands as a critical aspect of maintaining network security. Machine learning, a hot topic in artificial intelligence research, has found applications across multiple fields. Therefore, machine learning can be used for malicious code detection to achieve automation and intelligence in detection. This article takes dynamic API call sequences as the research object and extracts and processes API sequences from two perspectives. It then uses machine learning and deep learning algorithms for model training. Through the modification of model parameters and optimization of the network structure, the detection accuracy of the model is improved, fully leveraging the advantages of using machine learning algorithms for malicious program detection.

API call sequences are one of the most important features in malicious code detection. This article analyzes the current research status of existing malicious code detection methods, especially those based on API sequences. In response to the limitations of existing research, enhancements are made in both feature extraction and model training. From different perspectives, two malicious code detection methods were implemented. The experimental results show that both methods can effectively detect malicious code, highlighting their important research significance. The main achievements of this article are as follows:

We examined API call sequences in malicious code that can characterize the behavioral characteristics of malicious code as text, and then used advanced text classification-related technologies to classify malicious code. The intimacy analysis method, based on the self-attention mechanism, is used to extract key information. The feature extraction model based on self-attention mechanism uses pre-training to more efficiently obtain semantic information about the context in API call sequences. In comparison to BERT, it has the advantage of significantly reducing the number of parameters, making the model lighter and facilitating faster training. Finally, a simple TextCNN model is incorporated for malicious code classification. The experimental results show that the proposed model outperforms the baseline model in detecting and classifying malicious code, achieving good results.

There is still room for improvement in this method, particularly by integrating dynamic methods to extract the behavior of malicious code during runtime for analysis. Additionally, the incorporation of API call sequences from both dynamic and static analyses could potentially enhance detection results.

References

1. Wadkar, M., Troia, F.D., Stamp, M.: Detecting malware evolution using support vector machines. Expert Syst. Appl. **143**, 113022.1-113022.10 (2020)
2. Natani, P., Vidyarthi, D.: Malware detection using API function frequency with ensemble based classifier. In: International Symposium on Security in Computing & Communication, pp. 378–388 (2013)
3. Han, W., Xue, J., Wang, Y., et al.: MalDAE: detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. Comput. Secur. **83**, 208–233 (2019)
4. Cha, S.K., Moraru, I., Jang, J., et al.: SplitScreen: enabling efficient, distributed malware detection. J. Commun. Netw. **13**(2), 187–200 (2011)
5. Malhotra, A., Bajaj, K.: A hybrid pattern based text mining approach for malware detection using DBScan. CSI Trans. ICT **4**(2–4), 1–9 (2016)
6. Karnik, A., Goswami, S., Guha, R.: Detecting obfuscated viruses using cosine similarity analysis. In: Asia International Conference on Modelling & Simulation, pp. 165–170. IEEE Computer Society (2007)
7. Kinable, J., Kostakis, O.: Malware classification based on call graph clustering. J. Comput. Virol. **7**(4), 233–245 (2011)
8. Darshan, S., Kumara, M., Jaidhar, C.D.: Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm. In: 2016 11th International Conference on Industrial and Information Systems (ICIIS), pp. 534–549 (2016)
9. Fang, Y., Zhang, W., Li, B., et al.: Semi-supervised malware clustering based on the weight of bytecode and API. IEEE Access **8**, 2313–2326 (2019)
10. Lecun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436 (2015)
11. Xiaofeng, L., Fangshuo, J., Xiao, Z., Baojiang, C., Shengwei, Y., Jing, S.: A malicious sample detection framework based on the combination of API sequence features and statistical features. J. Tsinghua Univ. (Nat. Sci. Ed.) **58**(05), 500–508 (2018)
12. Cui, Z., Xue, F., Cai, X., et al.: Detection of malicious code variants based on deep learning. IEEE Trans. Ind. Inf. **14**, 3187–3196 (2018)