Copyright ⓒ 2024 KSII

MalEXLNet:A semantic analysis and detection method of malware API sequence based on EXLNet model

Xuedong Mao[1], Yuntao Zhao[1*], Yongxin Feng[2], and Yutao Hu[1]
[1] School of information science and engineering, Shenyang Ligong University
6 Nanping Middle Road, Hunnan District, Shenyang, Liaoning Province, 110159, China
[e-mail: maoxuedong2024@163.com, zhaoyuntao_2014@163.com, huyutao_2023@163.com]
[2] Graduate School, Shenyang Ligong University
6 Nanping Middle Road, Hunnan District, Shenyang, Liaoning Province, 110159, China
[e-mail: fengyongxin@263.net]
*Corresponding author: Yuntao Zhao

Received July 10, 2024; revised August 10, 2024; revised August 23, 2024; accepted August 31, 2024; published October 31, 2024

Abstract

With the continuous advancements in malicious code polymorphism and obfuscation techniques, the performance of traditional machine learning-based detection methods for malware variant detection has gradually declined. Additionally, conventional pre-trained models could adequately capture the contextual semantic information of malicious code and appropriately represent polysemous words. To enhance the efficiency of malware variant detection, this paper proposes the MalEXLNet intelligent semantic analysis and detection architecture for malware. This architecture leverages malware API call sequences and employs an improved pre-training model for semantic vector representation, effectively utilizing the semantic information of API call sequences. It constructs a hybrid deep learning model, CBAM+AttentionBiLSTM, for training and classification prediction. Furthermore, incorporating the KMeansSMOTE algorithm achieves balanced processing of small sample data, ensuring the model maintains robust performance in detecting malicious variants from rare malware families. Comparative experiments on generalized datasets, Ember and Catak, the results show that the proposed MalEXLNet architecture achieves excellent performance in malware classification and detection tasks, with accuracies of 98.85% and 94.46% in the two datasets, and macro-averaged and micro-averaged metrics exceeding 98% and 92%, respectively.

Keywords: Cyberspace security, Malware variant detection, XLNet, CBAM, AttentionBiLSTM

1. Introduction

The threat of malware has existed since the inception of computers. As security analysts and researchers continually improve defences, malware developers persist in innovating, discovering new infection vectors and enhancing their obfuscation techniques. Malware threats have continued to expand both vertically (i.e., in number and size) and horizontally (i.e., in type and function) due to the opportunities presented by technological advancements. The Internet, social networks, smartphones, IoT devices, and other technologies have facilitated the creation of intelligent and sophisticated malware [1]. According to statistics, thousands of new malware variants are developed and spread daily in cyberspace. Most of these malware variants are derived from mutations of known malware, such as new malware created from old versions through variations and polymorphisms. These new versions can alter their structure and functionality flow to evade antivirus software detection [2].

According to the AV-Test Institute, as of June 2024, a total of 883,810,844 Windows malware and 189,779,307 Windows PUAs have been discovered, representing increases of 44,439,528 and 661,824, respectively, compared to the previous year [3]. The Cyber Threat Report published by SonicWall indicates that SonicWall Capture Labs threat researchers recorded 6.06 billion malware attacks in 2023, an 11% year-over-year increase, marking the highest number of attacks since 2019 [4]. Data from the Kaspersky Security Bulletin 2023 shows that during the reporting period from November 2022 through October 2023, 437,414,681 malware attacks were thwarted from global online sources; financial malware was prevented from launching on 325,225 users' computers; more than 23,364 revised ransomware variants were discovered, along with 43 new ransomware families [5]. Therefore, there is a pressing need to design an effective automatic detection method against malware attacks.

There are two main approaches to malware detection. One is the static signature-based method, where the static characteristics of malware are stored in a database, and a file’s unique signature is compared against this database to determine if it is malware. While this method is the most common and convenient, it cannot recognise unknown malware, as only known variants are stored [6]. The other approach is behaviour-based detection, which examines a file’s behaviour and characteristics to identify whether it is malware. If confirmed, this method also classifies the malware family. Though more complex, behaviour-based detection yields better results for detecting and classifying unknown malware [1].

Since malware variants are updated and iterated at an increasing rate, identifying unlogged malware has become a top priority in malware detection. Although variant malware may exhibit different code sequences in various environments, it must maintain consistent behaviour across all environments. Since malware is designed to perform specific malicious activities, most detection and classification methods focus on behavioural characteristics rather than structural features. These methods use data such as Windows API calls, DNS parsing, and registry operations to reflect malware behaviour [7]. API calls are widely used for dynamic behavior-based analysis [8]. Sequences of API calls are considered representative of understanding malware's behavioral characteristics [9].

The development of artificial intelligence technology has simplified life by providing efficient solutions in different fields, including cybersecurity [10]. Malware variant detection using API call sequences can be achieved through machine learning and deep learning. However, machine learning faces limitations, including delays introduced by feature engineering and the need for extensive data preprocessing, which hinder real-time analysis. Adding data engineering layers to manage the growing data volume exacerbates these delays.

As a result, deep learning has been employed for malware detection, offering automated feature engineering, the ability to handle large datasets, extract features from limited data samples, and support one-shot learning [11].

However, existing deep learning-based research has limitations. Models like CNNs, RNNs, LSTMs, and BiLSTMs have been widely used in recent years to acquire sequence features and identify malicious behaviours automatically. Recent studies [12][13] found that these models can be spoofed through wrapping and black-box attack techniques. Two main issues contribute to this: first, simply mapping APIs to numeric values overlooks the inherent semantic features of functions; second, these models need to capture sequential features effectively. Additionally, they need help with large datasets containing numerous API types, extensive feature sets, and long sequences, which degrades performance.

The API call sequence of a program serves as context representing the program, and malware from the same family generally exhibits similar behaviour. Thus, the contextual semantic relationships of API call sequences are often alike [14]. The sequence context and implicit function semantics are crucial in API call sequence categorization. API function names imply various semantics, such as read, write, search, and download operations and related resources like system permissions, networks, registries, and graphical user interfaces. Encoding API call sequences with one-hot vectors produces high-dimensional vectors and leads to the loss of critical semantic information [15]. Techniques from natural language processing, such as text sequence processing [16] and word embedding [17], aid in dimensionality reduction and semantic representation.

Existing studies often use traditional word embedding methods, such as Word2vec, to map API sequences to high-dimensional word vectors [18]. Transformer-based pre-training models have shown superior performance over traditional models in analyzing the semantics of malware API call sequences. However, these models must fully exploit the semantic information in API call sequences [19]. To address this, we propose the MalEXLNet architecture focused on semantic analysis for malware detection.

The main contributions of this paper are as follows:1) A MalEXLNet model for malware detection oriented towards semantic analysis is introduced. This model is designed to leverage malware semantic information and sequence characteristics. Compared to traditional models, MalEXLNet more effectively captures behavioural characteristics across global API call sequences using the XLNet-based substitution language model;2) Due to differences between API call sequences and the pre-trained model corpus, directly using large language models is less effective for extracting semantic relationships. This paper proposes replacing XLNet's word splitter with an embedding layer, transforming API sequences into word vectors, which are then processed by XLNet's stacked Transformer-XL architecture for better semantic extraction, producing a feature vector matrix rich in malware behaviour information; 3) To address malware evasion techniques such as polymorphism and obfuscation, this paper proposes an ensemble model integrating CBAM and AttentionBiLSTM. CBAM extracts local features, while BiLSTM focuses on global patterns. An adaptive attention mechanism highlights critical features, and the final output is classified into malware families using Softmax.

The rest of the paper is organized: Section 2 reviews related work on malware variant detection. Section 3 proposes the MalEXLNet model for semantic-based malware variant detection and details its implementation. Section 4 compares the model's performance with baseline models, evaluating it based on word vectors and the hybrid deep learning approach. Section 5 presents the conclusions.

2. Related Works

Currently, most malware API classification research employs language or deep learning models. Deep learning models directly vectorize API function sequences and output classification results after training. Alternatively, language models map API call sequences to high-dimensional vectors using the extracted semantic information for malware classification.

This section introduces related research on using language models for semantic vectorization and feature extraction of API call sequences, including the techniques used and their advantages and disadvantages, as shown in **Table 1**.

Table 1. Advances in Semantic Analysis of API Sequences

Author	Critical Technologies	Benefit	Shortcoming
Zhang et al	The model based on API-Sequence-Semantic Fusion (Mal-ASSF) [15]	Mal-ASSF outperforms existing solutions by 3% to 5% in detection accuracy.	Further studies can be conducted to obtain a better representation of semantic features using pre-trained models.
Zhang et al	Skip-Gram+CNNs-BiGRU [20]	Reaching an accuracy of 0.9828 and an F1-Score of 0.9827.	Encoding API names manually suffers from a lack of flexibility.
Zhao et al	Semantic chain+Gated CNN+Bi-LSTM+Attention [21]	Obtain semantic chains by deconstructing the API and employing the parameters of the API to augment the semantic information	Does not fundamentally address the problem of concept drift.
Maniriho et al	Embedding+CNNs +BiGRU [22]	It achieved an F1-score of 0.99 on the training set and 0.98 on the unseen data.	Embedding cannot understand contextual semantic information.
Aggarwal et al	RF+(ELMo+Word 2Vec+BERT) [23]	It achieved an accuracy between 0.91 and 0.93.	Experimental data set too small.
Γιαπαντζής	XLCNN [24]	Semantic feature extraction using improved XLNet model	Experimental data set too small.
Liu et al	BERT+CNN-LSTM [18]	It achieved an accuracy of 98.81%.	Experimental data set too small.

Based on the contributions of the researchers above, the use of semantic information in existing studies on malware API call sequences is inadequate. Most studies still employ traditional methods, such as embedding, which must fully leverage these sequences' contextual information. Additionally, models that achieve better results are often applied to smaller datasets, which can lead to overfitting. This paper proposes a method that improves pre-training models specifically for malware API sequences and integrates them with a hybrid deep-learning model. This approach effectively addresses these issues and is validated on large datasets to ensure fair and reliable experimentation.

3. Methodology

This chapter presents the construction of the MalEXLNet architecture for the intelligent detection of malware variants based on semantic analysis, as proposed in this paper.

3.1 System overview

In this section, we introduce MalEXLNet, an intelligent architecture for malware semantic analysis and detection. This method is based on malware API sequences and utilizes an enhanced EXLNet model for semantic feature extraction and vectorized representation. The extracted semantic feature matrix is input to a hybrid deep learning model, CBAM+AttentionBiLSTM. Features captured by CBAM's localized feature extraction and AttentionBiLSTM's global long-term dependency capture are passed to a fully connected layer. The Softmax function classifies each API function call sequence into distinct malware families. The design of this method is illustrated in **Fig. 1**.

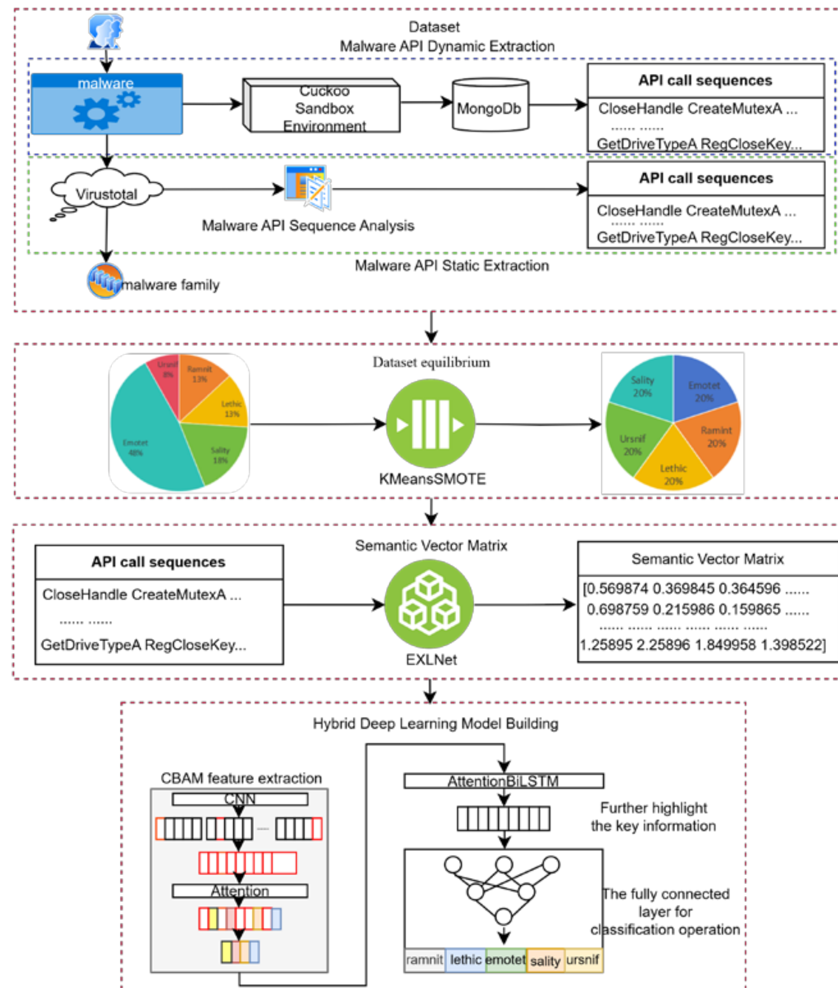

Fig. 1. MalEXLNet architecture

3.2 Generating API calls dataset

This section presents the construction of the Windows malware datasets Ember [25] and Catak [7], which are derived from dynamic and static analysis, respectively. The static Windows malware API dataset is constructed by analyzing only the PE file of the malware. In contrast, the dynamic Windows malware dataset requires analyzing the malware by executing it in a virtual sandbox. The detailed process of constructing these malware datasets is

illustrated in Fig. 2.

The static malware API dataset is generated based on the EMBER dataset. The EMBER dataset comprises millions of PE files representing various malware families and their variants. Each analyzed file in the EMBER dataset is stored in JSON format, where each entry corresponds to an analysis report of either benign or malicious malware. Each report includes a unique identifier (the SHA-256 hash and MD5 code of the original file), coarse time information (monthly resolution indicating when the file was first observed), a label (0 for benign, 1 for malicious, and -1 for untagged), and eight sets of raw characteristics, including histograms of parsed values, format-independent histograms, file properties, and import and export functions. The dataset is cleaned and anonymized to ensure it does not pose a security risk; the import functions contain API call sequences representing malware behaviour. The dataset is constructed by screening for malware in the EMBER dataset with tags other than 0, extracting their MD5 codes, and analyzing them using the VirusTotal online tool to determine each malware's family name. The tags are then reconstructed, and the API call sequences from the import table in each malware's analysis report are extracted to represent the malware's behavioural characteristics. The final dataset includes each malware's family label and its corresponding API function call sequence. The primary information of the EMBER dataset used in this study is summarized in Table 2.

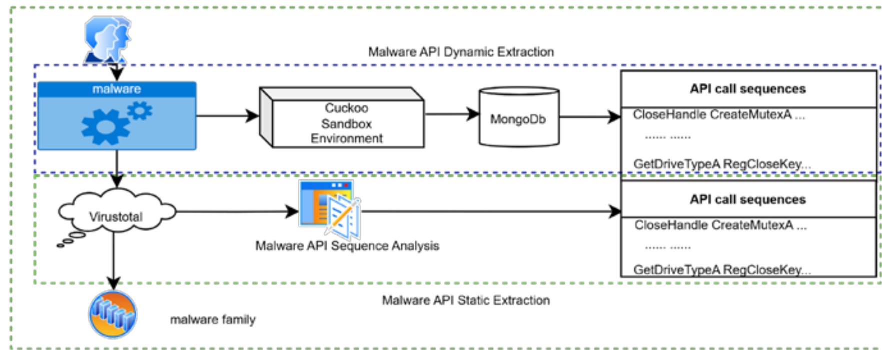

Fig. 2. Malware dataset construction process

Table 2. Main information of the EMBER dataset

Family	Number	Function	Label
Ramint	386	Multiple ways to spread, steal sensitive data, and remotely control infected devices.	0
Lethic	382	Sending large amounts of spam through infected computing devices with botnet characteristics that allow attackers to remotely control infected devices for illegal activities.	1
Emotet	527	Capable of stealing sensitive information, spreading other malware, and conducting large-scale attacks via botnets.	2
Sality	1413	Capable of spreading through infected executables and network shares, with the ability to steal sensitive information, download and execute other malware, and remotely control through the botnets they form.	3
Ursnif	241	Trojans that are mainly used to steal banking credentials, login information and other sensitive data, with features such as keylogging, screen capture and browser injection, and are often spread through spam and phishing attacks.	4
Total	2949		

The dynamic malware API dataset is created based on the Catak dataset. As illustrated in Fig. 2, this dataset must be produced within a virtual environment to avoid affecting the host computer. First, install the Ubuntu operating system and set up the Cuckoo Sandbox environment. Execute the malware within the Cuckoo Sandbox and write the resulting files into MongoDB. Analyze these files to generate the malware Windows API dataset. Table 3 presents critical information about the dynamic malware Windows API dataset constructed from the Catak dataset.

Table 3. Main information of the Catak dataset

Family	Number	Function	Label
Spyware	832	A type of malware that monitors and records user activity to steal personal information and sensitive data.	0
Virus	1001	Malware that spreads by infecting and modifying legitimate files typically disrupts system functionality or steals information.	1
Backdoor	1001	Malware that creates secret access routes in infected systems, enabling attackers to remotely control the system.	2
Downloader	1001	Programs used to download and execute other malware from the Internet are usually the first step in the attack chain.	3
Trojan	1001	Malicious programs that masquerade as legitimate software and are used to perform unauthorized operations, such as stealing data or controlling systems.	4
Adware	379	Malware that generates revenue by displaying ads or redirecting browsers typically impacts the user experience.	5
Worms	1001	Malware that replicates and spreads itself without user intervention and is commonly used for rapid proliferation and cyber attacks.	6
Dropper	891	Programs used to stealthily install and unleash other malware on infected systems usually avoid detection.	7
Total	7107		

3.3 Dataset equilibrium

Producing a balanced dataset is challenging because the number of malware samples from different families varies significantly, leading to an imbalanced dataset. Table 1 and 2 show that this imbalance can adversely affect the model's classification performance. Specifically, the model may become biased toward the larger sample groups, while its accuracy diminishes for smaller groups. This imbalance can result in lower prediction accuracy for less-represented malware families.

Data equalization algorithms balance the dataset to address these issues, ensuring that each category has approximately equal samples. After reviewing extensive literature and building on previous work [26][27][28], we initially selected the KMeansSMOTE, ADASYN, and Smote-EnN algorithms as potential methods. After comparing their performance on the dataset, as shown in Fig. 3, we decided to use KMeansSMOTE, ADASYN, and Smote-EnN for data equalization in this study.

The KMeansSMOTE algorithm effectively captures the distribution information of the original samples by partitioning the sample space into clusters and generating new samples within each cluster. This approach creates more representative synthetic samples and reduces overfitting to the overall sample distribution. Additionally, the algorithm accounts for the local

characteristics of each cluster, resulting in synthetic samples that are more similar to the original samples and minimizing noise. Finally, the algorithm handles nonlinear data structures well, offering improved oversampling results. The equalization results of the KMeansSMOTE, ADASYN, and Smote-EnN algorithms on the EMBER and Catak datasets are presented in **Fig. 3** and **4**.

3.4 API calls sequence feature-vectorized representation

Maniriho et al. [22] noted that similarities in API function call sequences differ significantly from those in ordinary English words or texts, leading to suboptimal results when using pre-trained language models directly. Consequently, they employed direct embedding to encode API function call sequences, enabling the automatic generation of dense embedding vectors. However, this vectorized representation fails to capture semantic relationships, making it challenging for the model to recognize shared behavioral patterns within the same family.

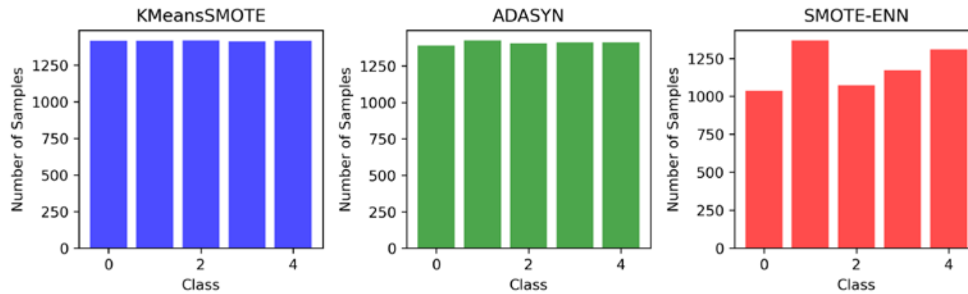

Fig. 3. Comparison of the effectiveness of equalization algorithms on the EMBER dataset

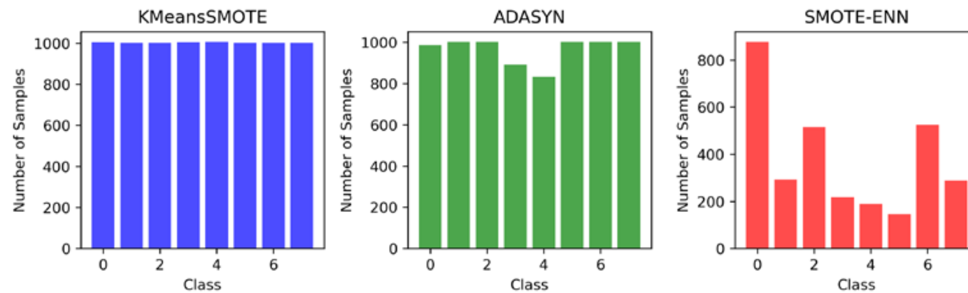

Fig. 4. Comparison of the effectiveness of equalization algorithms on the Catak dataset

Liu et al. [18] propose using the BERT pre-training model to vectorize the semantic features of API call sequences but do not address the inconsistency between API sequences and the pre-training corpus. Despite BERT's robust performance, the authors of XLNet [29] demonstrate that XLNet often surpasses BERT in various tasks and addresses some of its shortcomings. Therefore, this paper proposes a more suitable vectorization method for representing API function call sequence features: the Embedding+XLNet model architecture.

3.4.1 Embedding

After the dataset is prepared, the API function call sequences must be transformed into a format from which the model can learn; this process is known as word embedding. The word embedding method proposed in this paper begins by using keras_Embedding to encode the API function call sequences and generate word vectors. Specifically, Embedding first constructs a collection of all API function call sequences in the dataset. Each sequence within this collection is then subjected to sequential integer encoding, where similar sequences are

mapped to similar integers, forming a dictionary. The result is an integer-encoded representation of each API function call sequence.

Meanwhile, the Keras embedding layer uses a deep neural network to generate dense vector representations of API function call sequences. It maps encoded API calls in the input sequence to dense vectors in a high-dimensional space, ensuring that similar API calls are positioned closer together. Specifically, the Keras embedding layer processes an input matrix of integer-encoded API function call sequences, where each row represents the sequence of API calls for a particular malware family. Each integer encoding is mapped to a fixed-length dense vector, learned through neural network training. The final output is a dense vector matrix representing the API function call sequences.

Critical parameters of the Keras embedding layer include the vocabulary size of the dataset's API sequences and determining the number of unique sequences. The dimensionality of the embedding vector determines the length of each dense vector mapped from input words. The input sequence length (input_length) parameter, typically set based on the maximum sequence length in the dataset, guides Keras in creating an appropriately sized embedding matrix. For parameter selection in the Keras embedding layer, the dataset's API sequence length distribution is first analyzed (**Fig. 5**). Most sequences fall below 250 in length, so input_length is set to 250. After defining input_length, determining the embedding vector dimension (output_dim) involves clustering the dense vectors output by the embedding layer. Visual clustering analysis, employing methods like KMeans [30] and t-SNE [31] for dimensionality reduction, assists in choosing an appropriate output_dim. t-SNE, known for preserving data point distances across high-dimensional to low-dimensional spaces, uses Gaussian distribution to measure similarity and achieve effective data visualization through relative entropy minimization between spaces.

In the context of the KMeans algorithm, $\left\| x_i - x_j \right\|^2$ represents the squared Euclidean distance, and σ_i denotes the distance between data point i. The associated variance y_i refers to point i in the lower-dimensional space.K-means, a standard clustering algorithm, is a method of initially partitioning data into K clusters. Each data point is allocated to the cluster centre that is the nearest distance away. Subsequently, the algorithm computes the mean of all data points within each cluster, thereby updating the cluster centres. These steps are iterated until either the cluster centers converge or a predefined number of iterations is reached. The objective of K-Means is to minimise the within-cluster sum-of-squares error, which is a measure of the sum of distances from each point in a cluster to its centre.

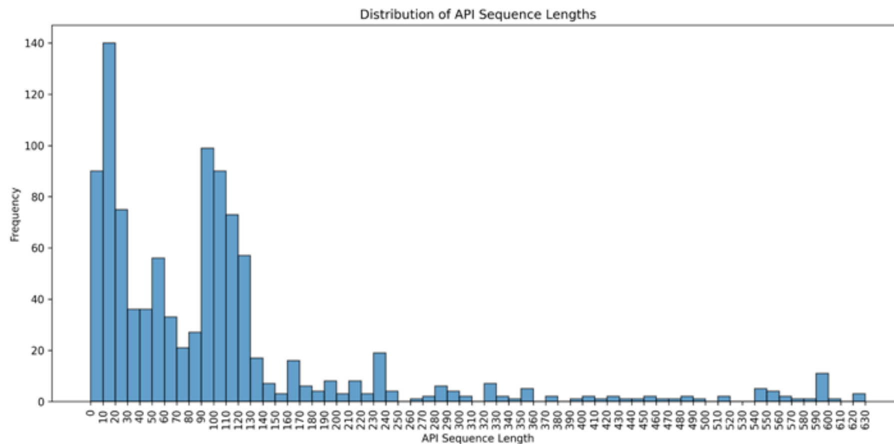

Fig. 5. Statistical plot of length distribution of API sequences

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\Sigma_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)} \quad (1)$$

$$q_{j|i} = \frac{exp(-||y_i - y_j||^2)}{\Sigma_{k \neq i} exp(-||y_i - y_k||^2)} \quad (2)$$

$$KL(p_{j|i}||q_{i|j}) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}} \quad (3)$$

Using the t-SNE and KMeans algorithms to visually cluster the word vectors generated by embedding, as shown in **Fig. 6**, we observe that clustering performance is optimal when output_dim is set to 50, maximizing feature extraction effectiveness.

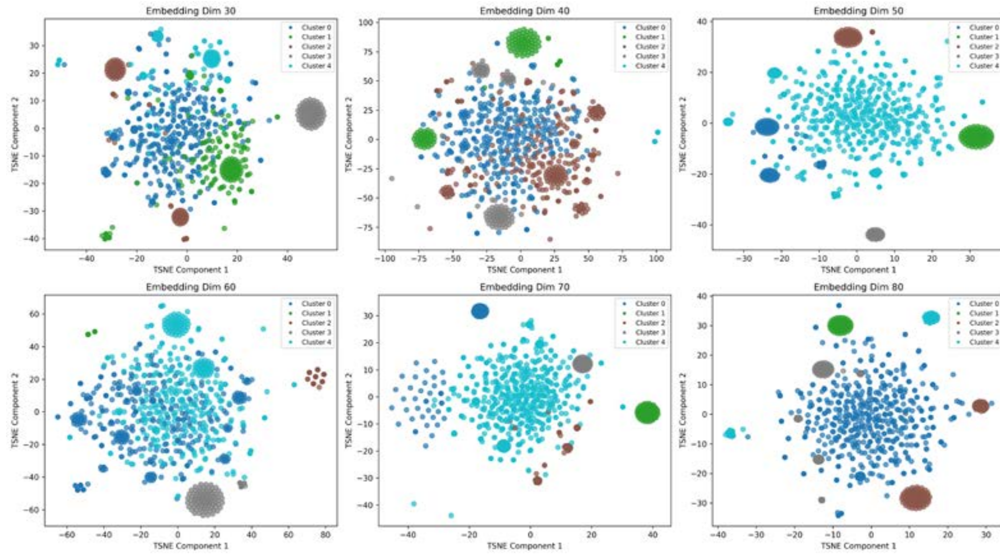

Fig. 6. Effect of output_dim clustering in different dimensions

We introduce the contour coefficient index to verify that the clustering effect is optimal when output_dim = 50. This index assesses the clarity of contours within each category after clustering. Its value ranges from -1 to 1, with higher values indicating better clustering effectiveness. The formula for calculating this coefficient is as follows:

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \quad (4)$$

In this context, a(i) represents the mean distance between the ith element x(i) and all other elements within the same cluster, indicating the degree of cohesion within the cluster. b(i) denotes the nearest mean distance between x(i) and elements of all other clusters, quantifying the dispersion between clusters. The silhouette coefficient is illustrated in **Fig. 7**.

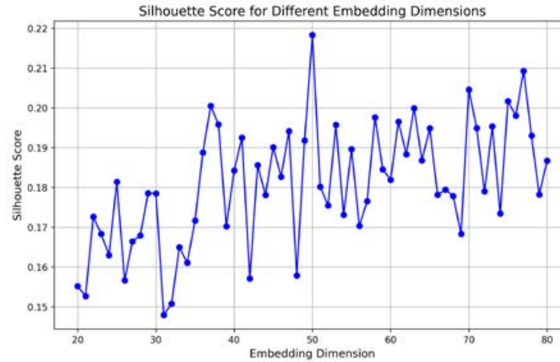

Fig. 7. Output _ dim contour coefficients of different dimensions

3.4.2 EXLNet model method

The XLNet model uses the SentencePiece segmentation method [32]. While SentencePiece effectively handles morphemes and unregistered words in regular text, API function call sequences often include many unique function names and identifiers, making subword segmentation less suitable. In contrast, embedding methods better preserve the integrity of function names and identifiers in API function call sequences. Additionally, embeddings allow for more flexible customization of sequence processing based on the specific characteristics of these sequences.

The word vectors obtained using only the Embedding model lack semantic relationships in the malware API call sequences. Therefore, the output of the Embedding layer is further processed with the pre-trained XLNet model to extract contextual semantic features from the API function call sequences, enriching the model's learnable features.

The XLNet model combines the strengths of autoregressive and autoencoding language models by introducing the bidirectional context modeling of autoencoding models into the autoregressive framework. This approach resolves the inconsistency between training and fine-tuning phases typically found in bidirectional autoencoding models due to [Mask]. Additionally, XLNet uses the Transformer-XL architecture to address the limitation of fixed-length input sequences in standard Transformer models [33]. Transformer-XL's extended contextual memory is particularly beneficial for handling long sequential tasks, allowing XLNet to excel in tasks requiring long-term dependency modeling.

The Two-stream in XLNet represents Two groups of information streams, called content stream and query stream, which are two hidden states, namely, the content hidden state $h_\theta(X_z < t)$ is abbreviated as h_{z_t} , and the query hidden state $g_\theta(X_z < t, z_t)$ is abbreviated as g_{z_t} . The query hidden state g_i^0 is initialized as a variable w, and the content hidden state is initialized as $h_i^0 = e(x_t)$, where $e(x_t)$ is the initial word vector.

Let m be the number of layers, and compute the query hidden state for each layer:

$$g_{z_t}^m \leftarrow Attn\left(Q = g_{z_t}^{m-1}, KV = g_{z \le t}^{m-1}; \theta\right) \tag{5}$$

Content hidden state:

$$h_{z_t}^m \leftarrow Attn\left(Q = h_{z_t}^{m-1}, KV = h_{z \leqslant t}^{m-1}; \theta\right) \tag{6}$$

In this way, the Query Stream can be used to predict the location without leaking the current location's content information, and the purpose of the autoregressive language model is achieved.

The Attention Score of q_i and k_i for the same Segment in a standard Transformer can be decomposed as follows.

$$A_{i,j}^{abs} = E_{x_i}^T W_q^T W_k E_{x_j} + E_{x_i}^T W_q^T W_k U_j + U_i^T W_q^T W_k E_{x_j} + U_i^T W_q^T W_k U_j \quad (7)$$

Based on the above formula, the Transformer-XL model proposes a calculation formula for relative position encoding Attention:

$$A_{i,j}^{rel} = E_{x_i}^T W_q^T W_{k,E} E_{x_j} + E_{x_i}^T W_q^T W_{k,R} R_{i-j} + u^T W_{k,E} E_{x_j} + v^T W_{k,R} R_{i-j} \quad (8)$$

The above two formulas contain four subformulas, numbered Formula A, Formula B, Formula C, and Formula D. Compared with Formula $A_{i,j}^{abs}$, the relative position code R_{i-j} in Formula $A_{i,j}^{rel}$ replaces the absolute position code U_j in Formula B and Formula D, and the trainable $u \in R^d$ and $v \in R^d$ are used to replace $U_i^T W_q^T$ in Formula C and Formula D, respectively.

Then, the key is split into $W_{k,E}$ and $W_{k,R}$ to represent the content and location-related keys. In Equation $A_{i,j}^{rel}$, equation a represents the content calculation, which is the Embedding of x_i times the inner product of the Embedding of transformation matrices W_q and x_i times $W_{k,E}$. Equation B represents the content-based position bias, the vector of i multiplied by the relative position encoding. C formula represents global content bias; Equation D represents the global position bias.

Then, the hidden state and relative position encoding obtained from the previous calculation are input into Transformer-XL, where they are used for computation. Transformer-XL enhances Transformer by adding a memory mechanism that saves previous hidden states for use in subsequent computations. Each Encoder in Transformer-XL includes a memory to store these states. When a new position is computed, the previous state is retrieved from memory and combined with the current input. The updated output is then stored in memory, extending the sequence length. The formula for vector concatenation is as follows:

$$H_\tau^{n-1} = [SG(m_\tau^{n-1} \sim o h_\tau^{n-1})] \quad (9)$$

The formula shows that the current input is memory (m_τ^{n-1}) and the hidden state h_τ^{n-1} at the previous moment. SG() means not participating in the gradient calculation, o means vector splicing, τ means the segment, and n means the number of layers.

Calculate the Query, Key, and Value:

$$q_\tau^n, k_\tau^n, v_\tau^n = h_\tau^{n-1} W_q^n, H_\tau^{n-1} W_{k,E}^n, H_\tau^{n-1} W_v^{nT} \quad (10)$$

In the formula, the query can only be calculated with the hidden state h_τ^{n-1} at the last time, and the Key and value are calculated with the H_τ^{n-1} in Equation 6. Because the Key is decomposed into $W_{k,E}$ and $W_{k,R}$, only the $W_{k,E}$ representing the content is used in the calculation here.

The Attention score is calculated from A^{rel}:

$$A_{\tau,ij}^n = q_{\tau,i}^n k_{\tau,j}^n + q_{\tau,i}^n W_{k,R}^n R_{i-j} + \mu^T k_{i,j}^n + v^T W_{k,R}^n R_{i-j} \quad (11)$$

Since E and W have been calculated to q, k and v in Eq. 6, they can be directly substituted here.

Convert the Attention scores into probabilities:

$$a_\tau^n = Mask - Softmax(A_\tau^n) v_\tau^n \quad (12)$$

The Softmax function converts the attention score into a number between 0 and 1; that is,

the attention score is converted into a probability.Residual join and layer normalization:

$$o_\tau^n = LayerNorm\left(Linear\left(a_\tau^n\right) + h_\tau^{n-1}\right) \tag{13}$$

Residual chaining enables the model to preserve information from the original input as it passes through, facilitating more efficient gradient propagation in deep networks. This capability allows for training deeper models without encountering vanishing or exploding gradients. Layer normalization standardizes features in each Transformer's self-attention mechanism and feedforward neural network. It helps reduce internal covariate shifts during training, thus stabilizing the training process. Layer normalization is often used with residual chaining to enhance the model's expressive power and training effectiveness.

Fully connected:

$$h_\tau^n = Positionwise - Feed - Forward\left(o_\tau^n\right) \tag{14}$$

Fully connected layers in each Transformer block help build deeper networks that more effectively capture features and dependencies in API sequences. After being processed by an XLNet model, the output feature matrix contains the contextual semantic relationships of the API sequences, providing richer features for the model to learn.

The Two-stream in XLNet represents Two groups of information streams, called content stream and query stream, which are two hidden states, namely, the content hidden state $h_\theta(X_z < t)$ is abbreviated as h_{z_t} , and the query hidden state $g_\theta(X_z < t, z_t)$ is abbreviated as g_{z_t} . The query hidden state g_i^0 is initiated as a variable w, while the content hidden state is initiated as $h_i^0 = e(x_t)$. The initial word vector is represented by $e(x_t)$.

The comparison of the relationship between using the embedding method, XLNet method, and the proposed method and the results after performing word vector transformation is shown in **Fig. 8**.

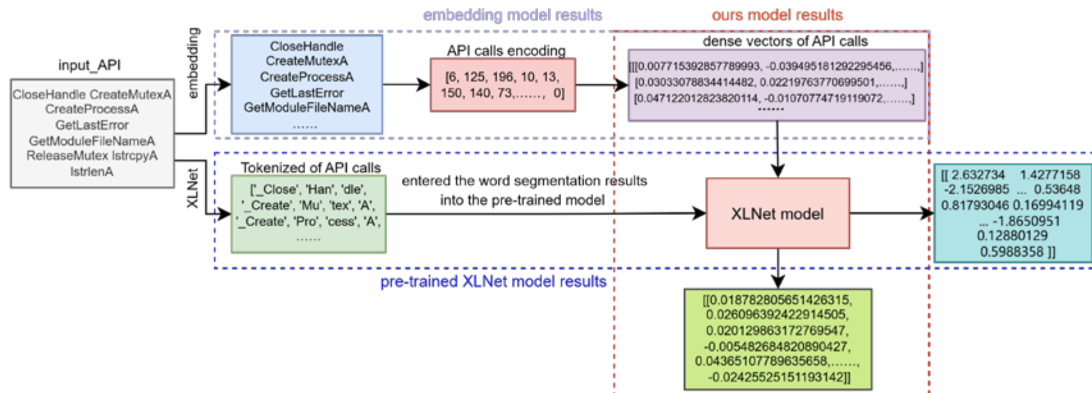

Fig. 8. Relationship between several word vector transformation methods and transformation result graph

3.5 Feature extraction for Hybrid deep learning models

We devised a hybrid deep learning model for automatic feature extraction to enhance the extraction of local semantic information from API sequences. This model integrates CBAM and AttentionBiLSTM, leveraging their strengths to effectively capture features from malware API call sequences and classify malware families.

3.5.1 CNN

Convolutional Neural Networks (CNNs) are prominent models in deep learning, typically comprising five layers: the input layer, convolutional layer, ReLU activation layer, pooling layer, and fully connected layer. One-dimensional convolution, a fundamental operation in CNNs, extracts local semantic information by sliding a small kernel (or filter) across the input feature matrix of API call sequences. This operation performs element-wise multiplication and summation to produce a new feature map at each position.

In our model, a one-dimensional CNN layer employs 64 convolutional kernels, each of width 3. ReLU activation is applied, and padding is used to handle boundary issues. Each convolutional layer is followed by a max pooling layer to reduce dimensionality and enhance feature extraction efficiency. This approach enables the model to capture local semantic features from API sequences better, facilitating more accurate classification and prediction in subsequent layers.

3.5.2 Convolutional Block Attention Mechanism CBAM

The Convolutional Block Attention Module (CBAM) is a streamlined enhancement for CNN. CBAM integrates spatial and channel attention mechanisms independently, sequentially inferring attention maps from intermediate feature maps. This process enhances feature optimization by multiplying the attention map with the input feature map. When applied to the analysis of malware API call sequences, CBAM effectively highlights crucial semantic information, significantly improving the accuracy of malware family classification. Refer to **Fig. 9** for the CBAM architecture diagram.

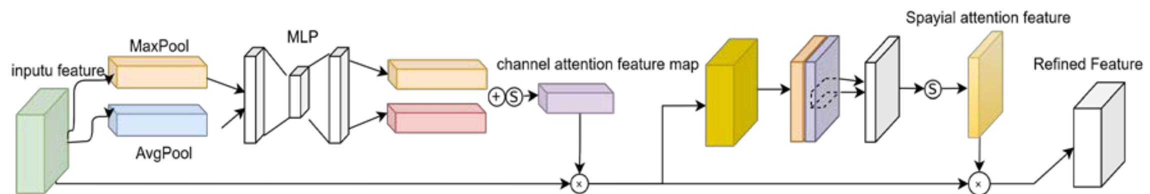

Fig. 9. Architecture diagram of CBAM

3.5.3 AttentionBiLSTM

After extracting local semantic features with CBAM, long-term dependencies are captured from a global perspective to provide a more comprehensive understanding of the API sequence's semantic information. This approach aims to capture bidirectional semantic information from API sequences while addressing the vanishing gradient problem, enhancing the model's ability to capture long-distance dependencies. The BiLSTM network consists of two LSTM layers: one processes the sequence in the forward direction and the other processes it in the reverse direction. Each LSTM layer includes a hidden state and a memory state. The input sequence is fed into the forward and reverse LSTM layers at each time step to compute hidden states in both directions. The structure of the LSTM is illustrated in **Fig. 10**.

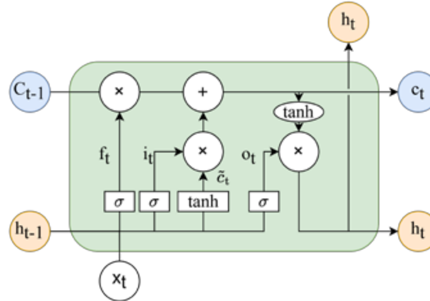

Fig. 10. LSTM structure

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{15}$$

$$i_t = \sigma(W_t[h_{t-1}, x_t] + b_i) \tag{16}$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \tag{17}$$

$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t \tag{18}$$

$$o_t = \sigma(W_0[h_{t-1}, x_t] + b_o) \tag{19}$$

$$h_t = o_t \times \tanh(c_t) \tag{20}$$

Where f_t is the forgetting state vector, W_f is the weight of the forgetting gate, h_{t-1} is the output vector state of the previous moment, x_t is the input vector of the hidden state connected to the input gate, b_f is the bias term to adjust the opening degree of the forgetting gate, it is the updated hidden state vector, tanh is the hyperbolic tangent function. $\tilde{c}_t$ is the updated state vector of the gating unit, c_t is the state vector of the updated gating unit, c_t is the state vector of the output gate, and ht is the state of the output gate vector of the final result.

An adaptive attention mechanism layer is incorporated into the model after the BiLSTM layer to enhance its ability to process the input sequence and identify critical information and dependencies. This mechanism typically consists of one or more parameterized attention heads with learnable weight parameters to compute attention scores for each time step in the input sequence. Each attention head generates a context vector by applying weights to aggregate the hidden states of the input sequence. The multi-head attention mechanism processes the input sequence in parallel, with each head learning different aspects of attention and concatenating its outputs.

The adaptive attention mechanism allows the model to assign weights to different parts of the sequence at various time steps, enabling automatic learning and adjustment of these weights. This capability helps capture critical information within the input sequence, enhancing sequence modeling performance.

The structure of AttentionBiLSTM is shown in **Fig. 11**.

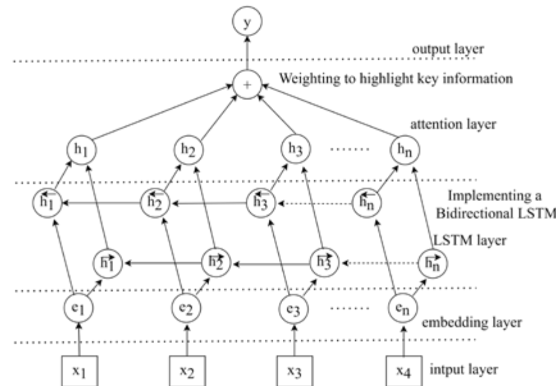

Fig. 11. Structure diagram of AttentionBiLSTM

3.6 Model Structure

In the preceding sections, the principles of each module in the model and their roles in the semantic analysis of malware API call sequences were described in detail. Initially, the malware API call sequences in the dataset underwent preliminary word embedding using an embedding method. The output from the embedding layer was then input into the XLNet model to extract and further enrich semantic relationships and optimize features. The embedding matrix, processed by both the embedding method and XLNet, was input to the hybrid deep learning model. After local feature extraction by CBAM, the AttentionBiLSTM captured long-term dependencies and highlighted key features. Finally, a softmax layer was employed for malware detection and classification.

The specific malware semantic analysis and detection model structure is depicted in **Fig. 12**.

4. Experiments and results

This section details the experimental evaluation performed in the evaluation of the proposed framework. A multi-classification problem has been generated based on a semantic analysis of malware API call sequences in Windows systems.

4.1 Experimental Setup

The malware static detection model proposed in this paper was implemented and tested in a Windows 10 (64-bit) computer with processor: 12th Gen Intel(R) Core(TM) i5-12490F 3.00 GHz, memory: capacity 16 GB, graphics card: NVIDIA GeForce RTX 4060 Ti 8GB; The Keras deep learning framework, versions TensorFlow 2.0.0 and Keras2.3.1, is used and the gradient descent optimization algorithm is Adam.

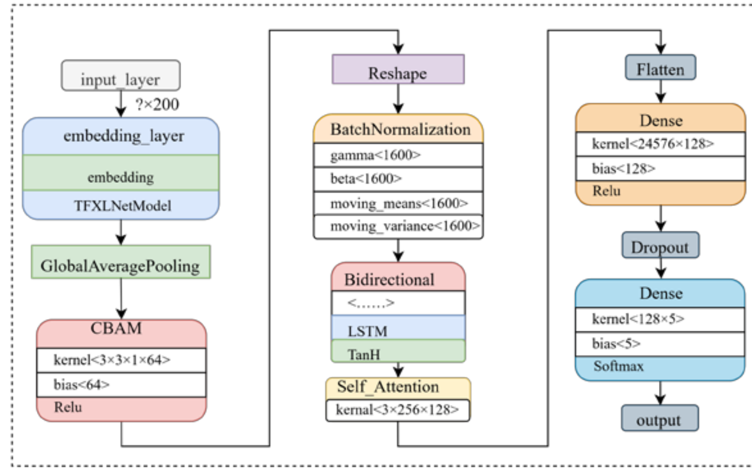

Fig. 12. Malware semantic analysis and detection model structure

4.2 Experimental parameter Settings

Table 4 provides a detailed account of the specific parameters associated with the proposed framework.

Table 4. Summary of parameter Settings for the proposed architecture

Layer	Parameter Used	Value
Embedding layer	Input sequence length Embedding output dimension Sequence padding	250 50 Zero padding
XLNet layer	Activation function Number of Hidden Units Dropout Probability Number of Hidden Layers Word Vector Dimension	GeLu 768 0.1 12 768
CBAM	Number of filters Kernel size Activation function Number of pooling layer Pooling method Stride	64 3 ReLu 2 Max pooling and Avg pooling 1
AttentionBiLSTM	Number of Hidden Units Activation function	128 Tanh
Fully connected layer	Number of hidden layers Number of neurons 1 Activation function L2 Regularizer Dropout Regularizer Number of neurons 2 Activation function	2 128 ReLu 0.01 0.5 5 Softmax
Model compilation	Optimizer Loss function	Adam with a learning rate of 1e-5 CategoricalCrossentropy
Others	Number of epochs Batch size	20/fold 16

4.3. Evaluation metrics

We used a suite of metrics commonly employed in classification problems to assess the model's efficacy, including accuracy, F1 score, and loss value. We selected the cross-entropy function as the model's loss function to quantify the discrepancy between the predicted and actual label distributions. Additionally, macro- and micro-averaging indicators were included.

4.4 Experimental Results

This section presents the results of the performance of the proposed model for the classification of malware API call sequences in the context of the detection task.

Fig. 13 illustrates the model's F1-score, accuracy, and loss values on the EMBER dataset. The experiment uses a 10-fold cross-validation method, with each fold representing a subset of the data. The results for each fold are shown as separate line plots. As depicted in the figure, the F1-score increases significantly with additional training and stabilizes after four training cycles. The model's training accuracy reaches 93% in the first cycle, with slight fluctuations in the subsequent three cycles, but remains above 98%. In later cycles, the model's performance stabilizes, with accuracy consistently around 98%. Initially, the model exhibits a high loss value, which decreases steadily over time, stabilizing at approximately 0.02 after four training cycles.

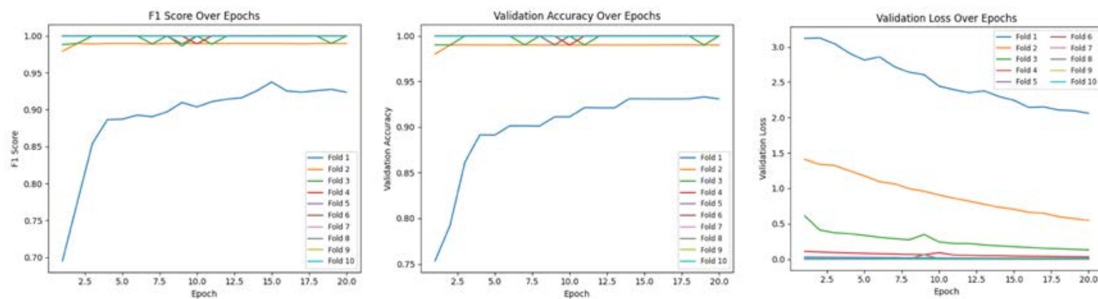

Fig. 13. MalEXLNet model test F1-score, Accuracy and Loss in EMBER

The experimental results of the model on the Catak dataset are presented in Fig. 14. The figure shows that both the accuracy and F1-score of the model increase gradually during the first round of training. From round 2 to round 4, the accuracy exhibits a slight decreasing trend, after which the model's performance gradually stabilizes. Ultimately, the F1-score, accuracy, and loss stabilize at approximately 97%, 98%, and 0.05, respectively.

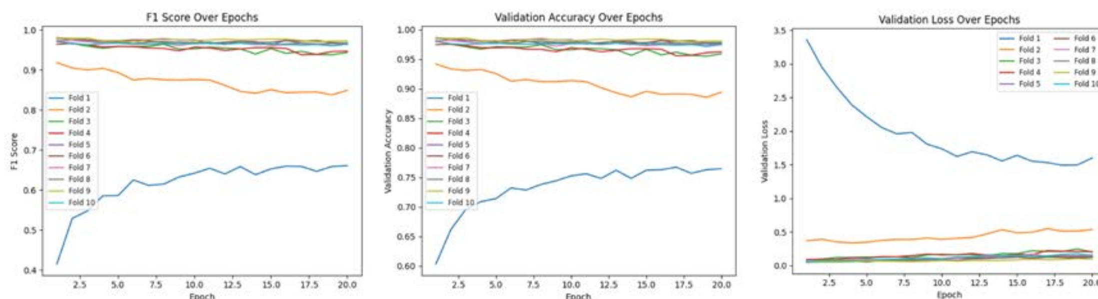

Fig. 14. MalEXLNet model test F1-score, Accuracy and Loss in Catak

4.5 Comparison with the baseline model

Table 5 and 6 present the macro- and micro-averaging metrics of the models on the EMBER and Catak datasets, along with comparative data against other baseline models. To ensure fairness, the parameters and experimental environments for the baseline models are kept consistent with those reported in the original studies. All models are evaluated using the dataset established in this paper. The macro- and micro-averaging metrics were also calculated for the baseline models to enhance the experiments' persuasiveness. These additions did not affect the models' performance.

The baseline models selected for comparison are as follows: Sahil et al. [23] used ELMo, Word2Vec, and BERT for semantic feature extraction of API sequences, achieving high accuracy of up to 93% on their dataset. Γιαπαντζής et al. [24] proposed an enhanced model named XLCNN, which combines the size and structure of feedforward neural networks with input dimensions to improve semantic analysis of malware and address malicious code classification. Liu et al. [18] introduced SeMalBERT, a semantic-based malware intelligence model that leverages the pre-trained BERT model to extract semantic relationships in API call sequences. They also vectorize these sequences and construct a CNN-LSTM classification model with an attention mechanism for malware classification and detection.

As illustrated in the preceding table, the proposed model achieves an accuracy of 98.85% on the test set, with an F1 score of 98.76%. The loss is minimal at 0.02, and macro and micro average indicators are above 98%. For comparison, other baseline models use deep learning methods for feature extraction. Among these, Γιαπαντζής et al. proposed an innovative word embedding processing method tailored to API sequence characteristics, achieving notable results. Liu et al. employed a hybrid deep learning model combining CNN and LSTM for feature extraction and classification, achieving better results than single deep learning models.

These two tables show that the model proposed in this paper achieves an accuracy of 98.85% on the EMBER dataset, with an F1-score of 98.76% and a loss value of 0.3882. Both macro-averaging and micro-averaging correlation indices exceed 98%. On the Catak dataset, the model achieves an accuracy of 94.46%, a loss value of 0.3287, and an F1-score of 92.13%. Other baseline models using deep learning methods for feature extraction also demonstrate notable performance. For instance, Γιαπαντζής et al. approached the problem with word vectors, proposing an innovative method tailored to API function call sequences, which yielded significant results. Liu et al. employed a hybrid deep learning approach, combining CNN and LSTM for feature extraction and classification, achieving superior results to single deep learning models.

Table 5. Comparison of different baseline models in EMBER

study	Acc	F1	Loss	macro			micro		
				P	R	F1	P	R	F1
Sahil [23]	0.9512	0.9485	0.6263	0.9502	0.9498	0.9425	0.9498	0.9462	0.9418
Γιαπαντζή ς[24]	0.9668	0.9635	0.5356	0.9638	0.9602	0.9597	0.9639	0.9613	0.9601
Liu[18]	0.9756	0.9704	0.4216	0.9712	0.9701	0.9609	0.9708	0.9716	0.9612
Proposed Model	**0.9885**	**0.9876**	**0.3822**	**0.9872**	**0.9862**	**0.9860**	**0.9841**	**0.9832**	**0.9827**

Table 6. Comparison of different baseline models in Catak

study	Acc	F1	Loss	macro			micro		
				P	R	F1	P	R	F1
Sahil [23]	0.8613	0.8523	0.8763	0.8465	0.8458	0.8460	0.8596	0.8436	0.8519
Γιαπαντζή ς[24]	0.8836	0.8759	0.7856	0.8657	0.8625	0.8653	0.8863	0.8769	0.8627
Liu[18]	0.9236	0.9216	0.5016	0.9136	0.9132	0.9135	0.9223	0.9165	0.9162
Proposed Model	**0.9446**	**0.9213**	**0.3287**	**0.9243**	**0.9210**	**0.9226**	**0.9446**	**0.9436**	**0.9448**

4.6 Performance comparison of different word embedding models

We conduct comparative trials, including those with traditional and emerging pre-trained language models, to show that our EXLNet model performs better in the semantic analysis of malware API call sequences.

Table 7 and 8 present the detailed experimental results of the three-word vector models on the EMBER and Catak datasets. The tables indicate that the XLNet model achieves the best performance in terms of accuracy and loss. Furthermore, the model proposed in this paper also attains superior results in both macro-averaging and micro-averaging evaluation metrics.

Table 7. Comparison results of different word vector models in EMBER

Model	Average			macro			micro		
	Acc	F1	Loss	P	R	F1	P	R	F1
Embedding	0.7978	0.7937	0.8780	0.8593	0.7930	0.823 6	0.7976	0.7946	0.7954
XLNet	0.9379	0.9340	0.1700	0.9400	0.9363	0.938 0	0.9356	0.9368	0.9362
EXLNet	**0.9620**	**0.9596**	**0.1092**	**0.9609**	**0.9603**	**0.960 6**	**0.9649**	**0.9627**	**0.9618**

Table 8. Comparison results of different word vector models in Catak

Model	Average			macro			micro		
	Acc	F1	Loss	P	R	F1	P	R	F1
Embedding	0.7978	0.7937	0.8780	0.8593	0.7930	0.8236	0.7976	0.7946	0.7954
XLNet	0.9379	0.9340	0.1700	0.9400	0.9363	0.9380	0.9356	0.9368	0.9362
EXLNet	**0.9620**	**0.9596**	**0.1092**	**0.9609**	**0.9603**	**0.9606**	**0.9649**	**0.9627**	**0.9618**

4.7 Ablation experiment

The experimental results in Table 9 and 10 demonstrate that our proposed model effectively leverages the strengths of each sub-module—the absence of any sub-module results in a noticeable degradation in the model's performance. For clarity, each model configuration is designated as Models 1 through 7, as outlined in Table 9.

Table 9. Results of ablation experiments in EMBER

Algorithm	Acc	F1	Loss	macro			micro		
				P	R	F1	P	R	F1
Dense	0.9580	0.9547	1.012	0.9589	0.9568	0.9578	0.9579	0.9567	0.9588
CNN	0.9623	0.9588	0.4320	0.9650	0.9623	0.9635	0.9612	0.9628	0.9635
CBAM	0.9713	0.9694	0.9627	0.9721	0.9705	0.9713	0.9704	0.9716	0.9728
BiLSTM	0.9810	0.9799	0.4712	0.9808	0.9803	0.9805	0.9806	0.9810	0.9808
Attention BiLSTM	0.9826	0.9814	0.4718	0.9826	0.9819	0.9822	0.9821	0.9828	0.9816
CBAM+ BiLSTM	0.9829	0.9817	0.4722	0.9831	0.9823	0.9827	0.9828	0.9831	0.9823
CBAM+ Attention BiLSTM	**0.9885**	**0.9876**	**0.3822**	**0.9872**	**0.9862**	**0.9860**	**0.9841**	**0.9832**	**0.9827**

In Model 1, the feature vectorization matrix derived from the semantic analysis of the EXLNet model is directly input into the fully connected layer, which performs classification and detection tasks. Model 2 extends Model 1 by incorporating a convolutional neural network (CNN) to extract local features. Comparing the results of Model 2 with Model 1 highlights the effectiveness of CNNs in local feature extraction. Model 3 builds on Model 2 by integrating spatial and channel attention mechanisms through the CBAM module. Results show that incorporating these attention mechanisms significantly enhances the model's feature extraction capabilities, ensuring critical features receive more attention. Model 4 advances beyond Model 1 by adding a bidirectional LSTM (BiLSTM) layer to filter and retain historical information, capture long-term dependencies, and improve the model's ability to learn semantic relationships in malware API call sequences. Model 5 further improves Model 4 with an adaptive attention mechanism, allowing the model to capture essential features better and enhance its learning capability. Models 6 and 7 combine advanced text feature extraction methods and achieve higher accuracy than the previous models.

Table 10. Results of ablation experiments in Catak

Algorithm	Acc	F1	Loss	macro			micro		
				P	R	F1	P	R	F1
Dense	0.8723	0.8698	1.582	0.8802	0.8765	0.8659	0.8712	0.8802	0.8698
CNN	0.9018	0.8965	1.2314	0.8996	0.8856	0.8849	0.8963	0.8954	0.9001
CBAM	0.9054	0.9038	0.9489	0.8979	0.8896	0.8893	0.8996	0.8941	0.9010
BiLSTM	0.9219	0.9220	0.6512	0.9208	0.9210	0.9118	0.9206	0.9118	0.9028
Attention BiLSTM	0.9326	0.9310	0.4738	0.9302	0.9330	0.9298	0.9287	0.9222	0.9279
CBAM+ BiLSTM	0.9429	0.9412	0.4136	0.9298	0.9396	0.9829	0.9402	0.9425	0.9436
CBAM+ Attention BiLSTM	**0.9446**	**0.9213**	**0.3287**	**0.9243**	**0.9210**	**0.9226**	**0.9446**	**0.9436**	**0.9448**

5. Conclusion and Outlook

This paper proposes an intelligent malware analysis and detection method based on semantic analysis. This method leverages the improved pre-training model XLNet and the

hybrid deep learning model CBAM+AttentionBiLSTM to accurately detect malware API call sequences. We first extract API function call sequences of malware using static and dynamic analysis techniques and construct a dataset by classifying the malware by family. Next, we perform semantic feature vectorization of the API sequences in the dataset using the word vector model XLNet and input the resulting feature vectors into the CBAM+AttentionBiLSTM model for training. Finally, we use the Softmax method to classify the malware families. The rationale and efficiency of the proposed model are validated through comparisons with various word embedding techniques and ablation experiments. The experimental results show that the MalXLNet method excels in malware detection, achieving higher accuracy than other baseline models and traditional methods and maintaining stable performance despite variations in software and system environments.

Despite our remarkable research results, some pressing issues remain:

1. The interpretability of the proposed models could be more straightforward, posing a significant challenge in deep learning. Future research should prioritize enhancing model interpretability and exploring methods to improve transparency and comprehensibility.

2. The extended training time challenges efficiency in practical applications. Therefore, future research should focus on developing lightweight models that reduce computational overhead while maintaining high accuracy.

3. Further research is needed to validate the model's robustness across different environments, thereby improving its generalization capabilities for practical use.

References

[1] D. Gibert, C. Mateu, and J. Planes, “The rise of machine learning for detection and classification of malware: Research developments, trends and challenges,” *Journal of Network and Computer Applications*, vol.153, Mar. 2020. Article (CrossRef Link)

[2] A. A. Al-Hashmi, F. A. Ghaleb, A. Al-Marghilani, A. E. Yahya, S. A. Ebad, M. Saqib, and A. A. Darem, “Deep-Ensemble and Multifaceted Behavioral Malware Variant Detection Model,” *IEEE Access*, vol.10, pp.42762-42777, Apr. 2022. Article (CrossRef Link)

[3] Malware Statistics and Trends Report, AV-TEST, Magdeburg, Germany, Mar. 4, 2024. [Online]. Available: https://portal.av-atlas.org/malware/statistics

[4] 2024 SonicWall Network Threat Report, SonicWall, Silicon Valley, California, USA, Jul. 25, 2024. [Oline]. Available: https://www.sonicwall.com/zh-cn/threat-report

[5] Kaspersky Security Bulletion 2023, Kaspersky, Moscow, Russia, Nov. 2023. Article (CrossRef Link)

[6] D. Ucci, L. Aniello, and R. Baldoni, “Survey of machine learning techniques for malware analysis,” *Computers & Security*, vol.81, pp.123-147, Mar. 2019. Article (CrossRef Link)

[7] F. O. Catak, A. F. Yazi, O. Elezaj, and J. Ahmed, “Deep learning based Sequential model for malware analysis using Windows exe API Calls,” *PeerJ Computer Science*, vol.6, Jul. 2020. Article (CrossRef Link)

[8] Z. Zhang, P. Qi, and W. Wang, “Dynamic Malware Analysis with Feature Engineering and Feature Learning,” in *Proc. of the AAAI Conference on Artificial Intelligence*, vol.34, no.01, pp.1210-1217, Apr. 2020. Article (CrossRef Link)

[9] E. Amer, I. Zelinka, “A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence,” *Computers & Security*, vol.92, May. 2020. Article (CrossRef Link)

[10] G. Munjal, B. Paul, and M. Kumar, “Application of Artificial Intelligence in Cybersecurity,” *Improving Security, Privacy, and Trust in Cloud Computing*, pp.127-146, 2024. Article (CrossRef Link)

[11] U. Tayyab, F. B. Khan, M. H. Durad, A. Khan, and Y. S. Lee, "A Survey of the Recent Trends in Deep Learning Based Malware Detection," *Journal of Cybersecurity and Privacy*, vol.2, no.4, pp.800-829, 2022. Article (CrossRef Link)

[12] W. Qiang, L. Yang, and H. Jin, "Efficient and Robust Malware Detection Based on Control Flow Traces Using Deep Neural Networks," *Computers & Security*, vol.122, Nov. 2022. Article (CrossRef Link)

[13] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, "Query-Efficient Black-Box Attack Against Sequence-Based Malware Classifiers," in *Proc. of ACSAC '20: Proceedings of the 36th Annual Computer Security Applications Conference*, pp.611-626, Austin, USA, Dec. 2020. Article (CrossRef Link)

[14] S. Aggarwal, "Malware Classification using API Call Information and Word Embeddings," *Master's Projects,* Ph.D. dissertation, Department of Computer Science, San Jose State University, 2023. Article (CrossRef Link)

[15] S. Zhang, J. Wu, M. Zhang, and W. Yang, "Dynamic Malware Analysis Based on API Sequence Semantic Fusion," *Applied Sciences*, vol.13, no.11, 2023. Article (CrossRef Link)

[16] Q. Wang, and Q. Qian, "Malicious code classification based on opcode sequences and textCNN network," *Journal of Information Security and Applications*, vol.67, 2022. Article (CrossRef Link)

[17] J. Kang, S. Jang, S. Li, Y. Jeong, and Y. Sung, "Long short-term memory-based Malware classification method for information security," *Computers & Electrical Engineering*, vol.77, pp.366-375, 2019. Article (CrossRef Link)

[18] J. Liu, Y. Zhao, Y. Feng, Y. Hu, and X. Ma, "Semalbert: Semantic-based malware detection with bidirectional encoder representations from transformers," *Journal of Information Security and Applications*, vol.80, 2024. Article (CrossRef Link)

[19] Y. Hua, Y. Du and D. He, "Classifying Packed Malware Represented as Control Flow Graphs using Deep Graph Convolutional Neural Network," in *Proc. of 2020 International Conference on Computer Engineering and Application (ICCEA)*, pp.254-258, 2020. Article (CrossRef Link)

[20] Y. Zhang, S. Yang, L. Xu, X. Li, and D. Zhao, "A Malware Detection Framework Based on Semantic Information of Behavioral Features," *Applied Sciences*, vol.13, no.22, 2023. Article (CrossRef Link)

[21] D. Zhao, H. Wang, L. Kou, Z. Li, and J. Zhang, "Dynamic Malware Detection Using Parameter-Augmented Semantic Chain," *Electronics*, vol.12, no.24, 2023. Article (CrossRef Link)

[22] P. Maniriho, A. N. Mahmood and M. J. M. Chowdhury, "API-MalDetect: Automated malware detection framework for windows based on API calls and deep learning techniques," *Journal of Network and Computer Applications*, vol.218, 2023. Article (CrossRef Link)

[23] S. Aggarwal, and F. D. Troia, "Malware Classification Using Dynamically Extracted API Call Embeddings," *Applied Sciences*, vol.14, no.13, 2024. Article (CrossRef Link)

[24] Γιαπαντζής, "XLCNN: pre-trained transformer model for malware detection," M.S. thesis, 2024. Article (CrossRef Link)

[25] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," *arXiv :1804.04637*, Apr. 2018. Article (CrossRef Link)

[26] L. Akritidis, and P. Bozanis, "A clustering-based resampling technique with cluster structure analysis for software defect detection in imbalanced datasets," *Information Sciences*, vol.674, Jul. 2024. Article (CrossRef Link)

[27] P. Mahalakshmi, V. Mahalakshmi, E.S. Vinothkumar, B. Senthilkumar, M. Dinesh, and R. Krishnaprasanna, "A Real-Time Spam Identification Scheme Over Social Networking Environment Using Deep Learning Principles," in *Proc. of 2023 3rd International Conference on Mobile Networks and Wireless Communications (ICMNWC)*, pp.1-6, 2023. Article (CrossRef Link)

[28] R. Han, K. Kim, B. Choi, and Y. Jeong, "A Study on Detection of Malicious Behavior Based on Host Process Data Using Machine Learning," *Applied Sciences*, vol.13, no.7, 2023. Article (CrossRef Link)

[29] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, Q. V. Le, “XLNet: Generalized Autoregressive Pretraining for Language Understanding,” in *Proc. of 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, Vancouver, Canada, 2019. Article (CrossRef Link)
[30] M. Ahmed, R. Seraj and S. M. S. Islam, “The k-means Algorithm: A Comprehensive Survey and Performance Evaluation,” *Electronics*, vol.9, no.8, Aug. 2020. Article (CrossRef Link)
[31] L. V. Maaten and G. Hinton, “Visualizing Data using t-SNE,” *Journal of Machine Learning Research*, vol.9, no.11, pp.2579-2605, Nov. 2008. Article (CrossRef Link)
[32] T. Kudo and J. Richardson, “Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing,” *arXiv:1808.06226*, Aug. 2018. Article (CrossRef Link)
[33] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le and R. Salakhutdinov, “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context,” *arXiv:1901.02860*, Jun. 2019. Article (CrossRef Link)

Xuedong Mao Born in Yingkou, Liaoning Province in 2000, he received his bachelor's degree in Communication Engineering from Shenyang Ligong University in Shenyang, China in 2022. He is pursuing a master's degree in Communication Engineering at the School of Information Science and Engineering at Shenyang Ligong University in Shenyang, China. During the study period, he won the first-class scholarship and won awards in many competitions. His research interests include cyberspace security and artificial intelligence.

Yuntao Zhao completed his Ph.D. in Navigation, Guidance, and Control from the Nanjing University of Science and Technology. He is currently a professor and doctoral supervisor at the School of Information Science and Engineering at Shenyang Ligong University. His main research areas are Cyberspace Security, Machine Learning, and Deep Learning Algorithms.

Yongxin Feng received an M.S. in computer science from Northeastern University in 2000 and a Ph.D. in computer science and technology from the School of Information Science and Engineering, Northeastern University, in 2003. She is currently a Professor at Shenyang Ligong University. She has authored over 60 papers in related international conferences and journals. Her research interests include network management, wireless sensor networks, and communication and information systems. She received the ICINIS 2011 Best Paper Awards and 15 Science and Technology Awards, including the National Science and Technology Progress Award and the Youth Science and Technology Awards from the China Ordnance Society.

Yutao Hu received his B.S. and M.S. degree in computer science and technology from the Shenyang University of Technology and the Shenyang Ligong University. He is currently working toward the Ph.D degree from the Shenyang Ligong Unviersity, Shenyang, China.His Ph.D. major research is the direction of cyber security, sensor networks and communication and information systems.