Dynamic Malware Detection based on Enhanced Semantic API Sequence Features

Zhiguo Chen[a,b], Lei Zhou[a,b], Qingcheng Liu[a,b], Weizhi Meng[c,*], Jian Weng[d]

[a]*Engineering Research Center of Digital Forensics, Ministry of Education, Nanjing University of Information Science and Technology, Nanjing 210044, China*
[b]*School of Computer Science and School of Cyber Science and Engineering, Nanjing University of Information Science and Technology, Nanjing 210044, China*
[c]*School of Computing and Communications, Lancaster University, LA1 4YW Lancaster, U.K.*
[d]*College of Cyber Security, Jinan University, Guangzhou 510632, China.*

Abstract

Dynamic malicious software detection aims to assess whether executable programs exhibit malicious behavior by thoroughly studying and analyzing their dynamic features. However, many current methodologies insufficiently explore the semantic features of API sequences and instead rely more on mining parameter information during API call processes to enhance detection performance. This leads to issues such as excessive dependence on prior knowledge, larger model parameter sizes, and higher computational complexities. To that end, this paper proposes an enhanced semantic API sequence feature dynamic malware detection scheme that integrates the RoBERTa pre-training model and gating mechanism. This scheme solely leverages API call sequences that can comprehensively capture the contextual semantic information implicitly embedded during executable file execution. Meanwhile, dynamically adjusting the weights of various modal features within the model enhances sensitivity to different malicious software samples. By fusing multidimensional features, our approach comprehensively captures both the semantic and global characteristics of API sequences, enabling the model to adapt more flexibly to malware variants and thereby improving detection

*Corresponding author
Email addresses: chenzhiguo@nuist.edu.cn (Zhiguo Chen), zhoulei@nuist.edu.cn (Lei Zhou), qingcheng_liu@nuist.edu.cn (Qingcheng Liu), weizhi.meng@ieee.org (Weizhi Meng), cryptjweng@gmail.com (Jian Weng)

accuracy and robustness. Extensive experiments on four publicly available datasets demonstrate that the proposed method consistently achieves higher detection accuracy and strong generalization across different datasets and task types, including both binary and multi-class classification, thereby validating its effectiveness and practical applicability in dynamic malware detection.

Keywords: Malware Detection, Dynamic Analysis, API Call Sequence, Pre-trained Models.

1. Introduction

With the rapid proliferation of computers and networks, the number and variety of malicious software are continuously increasing. Each day, newly emerging zero-day malware poses a substantial threat to users' property, personal information, and device security. According to the latest data from AV-TEST [1], in 2023 alone, 104,857,417 new malicious software samples were recorded, with the weekly count reaching the magnitude of millions. Therefore, the timely detection of malicious software and its emerging variants is crucial for maintaining system and network security [2].

Researchers have proposed various static and dynamic analysis methods [3] to detect the exponentially growing malicious software. Static analysis, which does not require executing the malicious software, involves studying the source code or binary files [4] to extract static features such as strings, function calls, and API calls for analysis. It can avoid the risk of infecting terminals by running malicious software and consumes fewer resources. However, malware developers employ various obfuscation techniques such as code encryption, program instruction reordering, and dead code insertion to evade malware analysis [5],[6],[7]. These techniques require static analysis methods to spend more time parsing executable file structures, thereby reducing the efficiency and accuracy of the analysis. Additionally, static analysis typically relies on known features or signatures, which limits its effectiveness against unknown or novel malicious software. To overcome these limitations, dynamic analysis involves loading executable programs into main memory to monitor their actual runtime behavior. Through analysis of their behavior, it has the capability to detect zero-day malicious software [8], [9].

On the other hand, dynamic analysis monitors behaviors such as registry key changes, URL accesses, and API calls during the execution of executable programs, and analyzes their behavioral patterns. Previous studies

have shown that malware families often exhibit identical or highly similar behaviors [10]. For instance, trojans like Zeus frequently invoke APIs such as InternetOpen and HttpSendRequest to communicate with remote servers and use RegCreateKeyEx and RegSetValueEx to modify the registry for persistence. In contrast, ransomware like WannaCry commonly calls APIs such as CreateFile and WriteFile to encrypt files and establish communication with command-and-control servers via socket and connect to transmit ransom demands.

Therefore, analyzing and studying runtime API call sequences to uncover executable behavior patterns has become a focal point of research in the field of dynamic malware identification and detection. Zhang et al. [11] employed Gated-CNNs to transform API and their parameters, constructing a classification system with an LSTM network model. Building upon Zhang et al.'s [11] methodology, Li et al. [12] applied embedding and convolutional layers to extract API sequences and their intrinsic features. Then, they further utilized a Bi-LSTM module to explore the relationship information among multidimensional features. Chen et al. [10] utilized clustering to label APIs based on API parameters and employed TextCNN to explore the contextual associations of API call sequences. Li et al. [13] constructed API invocation graphs and applied graph neural networks (GNNs) to classify malicious software based on the analysis of API call sequences and parameters. Cui et al. [14] designed temporal process graphs (TPGs) and temporal API graphs (TAGs) to simulate inter-process and intra-process behaviors based on API and inter-process call relationships. Subsequently, they constructed a classification system based on behavioral graphs. Thus, methods that extract behavioral patterns by analyzing API call sequences have been proven effective in detecting and classifying malicious software [11], [12], [10], [13], [14]. However, many existing methods rely on parameter information during API calls to build detection and classification systems, resulting in several issues: (1) overreliance on prior knowledge, (2) large model parameter volumes, (3) high computational complexity, (4) an inability to fully capture complex semantics due to ignoring the dynamic order of API calls in context.

Motivations. To address the issues of insufficient semantic mining of API sequences in existing research and the high complexity of API call parameters, which make malware identification and detection difficult to implement in practice, this paper proposes a malware detection and classification scheme based on a pre-trained model and enhanced semantic API sequence features. This scheme dynamically adjusts the feature weights between se-

mantics of API sequences and global dimensions via a gating mechanism, aiming to deeply mine multidimensional features that can enhance the efficiency of detecting malicious behaviors in executable programs.

Currently, BERT [15], RoBERTa [16] and other large pre-trained language models (PLMs) based on Transformer have demonstrated favorable performance in tasks such as relation classification [17] and sentiment analysis [18] in natural language processing, as well as in fields like computer vision [19]. The multi-head self-attention mechanism of Transformer allows each word to attend to all preceding words or every word except the target, enabling the model to efficiently capture long-range dependencies without the expensive recurrent computations found in LSTMs [20]. As the potential features in API calls also describe behaviors or events in a sequential logical relationship, Xu et al. [21] and Demirkıran et al. [22] have successfully applied the BERT model to malware detection and classification tasks to extract semantic features from API sequences. Although Transformer-based models can consider the contextual information of each position in the input sequence simultaneously, the aforementioned studies still have some limitations: (1) They only perform simple fine-tuning on pre-trained models without deeply exploring the structural information of different malware API call sequences, which may limit the generalization ability of the approach; (2) They use the natural language vocabulary in pre-trained models to segment API names, which may lead to unexpected splits of critical API names (e.g., "NtCreateFile" being segmented into ["Nt", "Create", "File"]), thereby compromising the integrity of API call information and negatively impacting model performance.

Contributions. To mitigate the fragmentation of API sequences caused by directly transferring PLMs to the field of malware detection, this paper constructs an API call dictionary and utilizes the RoBERTa model for the first time to process API sequences as a whole, capturing the dynamic behavior of the entire executable. Furthermore, to compensate for the information loss due to not utilizing API call parameters, this paper employs a gating mechanism in conjunction with the global information of API sequences to construct enhanced semantic API sequence features, thereby improving the scheme's capability for malware detection and classification.

- We propose a method that utilizes the RoBERTa model to extract semantic information between API call sequences, thereby enhancing the model's ability to capture potential relationships between API calls.

- We propose a gating mechanism that dynamically adjusts the relative weights of global features and contextual semantic information in executable program API sequences, enabling the construction of enhanced semantic representations that improve detection sensitivity across diverse malware samples.

- Through comprehensive evaluations across multiple datasets and detection scenarios, the results demonstrate that the proposed semantic modeling and feature regulation mechanisms can effectively adapt to diverse data distributions, exhibiting strong transferability and generalization potential, and are therefore suitable for cross-scenario malware detection.

In the rest of this paper, we organize it as follows. We start by reviewing related work in Section 2, providing context for the research status. Our core contributions are discussed in Section 3, where we detail our proposed scheme. Section 4.1 evaluates its performance and discusses the limitations and future directions in this field. Finally, in Section 6, we draw our conclusion.

2. Related Work

Dynamic analysis methods require executing test programs in a simulated environment (e.g., sandbox) to capture process information during program execution and determine whether they exhibit malicious behavior by analyzing API calls, network communications, and other relevant features. API calls, in particular, provide crucial information for identifying anomalies and malicious activities. This section will focus on malware identification methods based on API calls, which can be broadly categorized into three main approaches: rule and pattern-based identification methods, machine learning-based identification methods, and deep learning-based identification methods.

2.1. Rule and pattern-based identification

Tian et al. [23] identified malicious software by utilizing a hash table to store the frequency of API strings. Hansen et al. [24] further converted the API calls and API frequency information of malicious software into feature vectors, classifying malicious software by computing the entropy of vector information. Additionally, Amer et al. [25] clustered functions with similar

API contextual features into the same cluster and utilized Markov chain algorithms to identify the behavior of API call sequences in dynamic analysis reports, constructing a malicious software detection system that achieved a classification accuracy of 99.7%.

Rule or pattern-based methods can effectively capture the structural information of API sequences in dynamic analysis reports of executable programs, thereby uncovering potential malicious patterns within programs. However, this approach relies on specific patterns within API sequences while disregarding the contextual information of API calls, potentially resulting in the loss of partial feature information.

2.2. Machine learning-based identification

Traditional machine learning algorithms extract and analyze features from API call sequences to obtain crucial information, enabling effective analysis, identification, and detection of program behavior. For instance, Singh et al. [26] employed the Shannon entropy of printable strings (PSI) and API calls, alongside registry modifications extracted from dynamic reports, as input features. They subsequently applied machine learning algorithms, including AdaBoost and Gradient Boosting (GB), for malware detection. Rabadi et al. [27] transformed API sequences and their parameters into hash representations, achieving classification accuracies of 99.2% and 99.4% using the XGBoost classification algorithm. Ndibanje et al. [28] calculated the similarity between API sequences and APIs in the Microsoft Developer Network (MSDN), employing the k-nearest neighbors (KNN) algorithm to identify and classify malicious software behavior. Zhang et al. [29] constructed a call relationship graph using API and parameter information from dynamic execution reports and built a malware classification system using clustering algorithms. Cui et al.[14] designed temporal process graphs (TPG) and temporal API graphs (TAG) to simulate inter-process and intra-process behaviors, proposing a heuristic random walk algorithm to explore representative malicious API call paths in the graph. Based on this, a classification scheme built using machine learning algorithms such as KNN achieved a classification accuracy of 99.23%.

Feature-based machine learning methods offer several advantages in malware detection, including high computational efficiency, interpretable models, and controllable features, which facilitate effective analysis and interpretation of malicious behavior. By extracting and selecting features from malicious

software, these methods can effectively identify patterns of malicious behavior. However, these approaches have drawbacks such as reliance on manually designed features, limited generalization capability, and high demand for domain expertise, which constrain their effectiveness and reliability when faced with complex malware and unknown malicious behavior patterns.

2.3. Deep learning-based identification

The malware detection methods based on deep learning offer advantages such as automatic feature learning, handling complex nonlinear relationships and large-scale data, and strong generalization capability. This approach enables better capturing patterns and relationships within data, thus effectively handling unknown malware samples and their variants. Zhang et al. [11] employed feature hashing techniques to encode APIs and their parameters, constructing a detection network using Gated Convolutional Neural Networks (Gated-CNNs) and Bidirectional Long Short-Term Memory networks (Bi-LSTMs) to identify malware. Xiaofeng et al. [30] utilized a Bidirectional Residual LSTM model and their proposed AMHARA algorithm to analyze the invocation information of APIs and their parameters, for constructing a malware classification system. Chen et al. [10] performed preliminary clustering analysis on APIs based on their parameters, and then employed the labeled API sequences obtained from clustering in conjunction with deep models for malware detection, achieving a detection accuracy of 98.52%. Li et al. [13] constructed API call relationships based on APIs and their parameters, implementing malware detection and classification tasks using graph neural networks.

The deep learning-based malware identification methods can flexibly explore the API and its parameter information from dynamic analysis reports, effectively capturing their intrinsic structure and patterns. However, these methods may face challenges such as complex feature processing, large model parameter sizes, and high computational complexity when detecting and classifying malware by exploring APIs and their invocation information. Meanwhile, Bi-LSTM encounters challenges in effectively memorizing and utilizing long-range dependencies as the length of the API sequence increases. The convolution operation of Text-CNN on local windows to extract API sequence features may result in the model struggling to capture comprehensive global semantic information. The local information propagation mechanism of GNN restricts the model's understanding of the overall structure of lengthy API

sequences. Hence, the aforementioned methods face difficulties in deeply exploring the global semantic information and underlying patterns of malicious behavior within lengthy API sequences.

Pre-trained models undergo pre-training on large-scale datasets, enabling them to understand the semantic rules within sequences comprehensively. Additionally, the model's self-attention mechanism dynamically captures semantic correlations at different positions within long sequences, thereby facilitating in-depth exploration of the global semantic information within the sequences. Similarly, the API call sequences in dynamic behaviors are also described with temporal structures, employing pre-training methods enables further exploration of potential global semantic information and behavioral patterns within API sequences without using API call parameters. For instance, Xu et al. [21] proposed the Malbert approach, utilizing the pre-trained model BERT for malware detection, achieving a detection accuracy of 99.9% on both the Ki and Catak datasets. Additionally, Demirkıran et al. [22] applied the pre-trained models BERT and CANINE for malware classification, demonstrating excellent performance in the classification of highly imbalanced malware families. The malicious software identification method based on BERT can capture the semantic correlations and contextual information between API calls. However, in the field of malware detection, BERT has the following issues: (1) Consecutive API call sequences often lack clear sentence boundaries. Using the Next Sentence Prediction (NSP) task may result in unnatural segmentation, thereby disrupting the continuity of overall semantic information. (2) The static masking mechanism limits the deep exploration of global semantics in long sequences.

This paper employs the RoBERTa model to extract semantic features from raw API sequences. RoBERTa, in contrast to BERT, eliminates the next sentence prediction task during pre-training, allowing it to process the API call sequence of a single malicious software as a whole, thereby avoiding the information loss and computational efficiency reduction issues associated with splitting API call sequences into multiple sentences. Additionally, RoBERTa employs dynamic masking, which aids the model in better understanding context, thereby more accurately capturing the semantic information in API sequences. Furthermore, to further capture API dependencies, we propose a gating mechanism to dynamically adjust the input weights of semantic and global features in the detection model, thereby forming semantically enhanced API sequence features to reduce the complexity of feature processing and model parameter volume, further enhancing the scheme's abil-

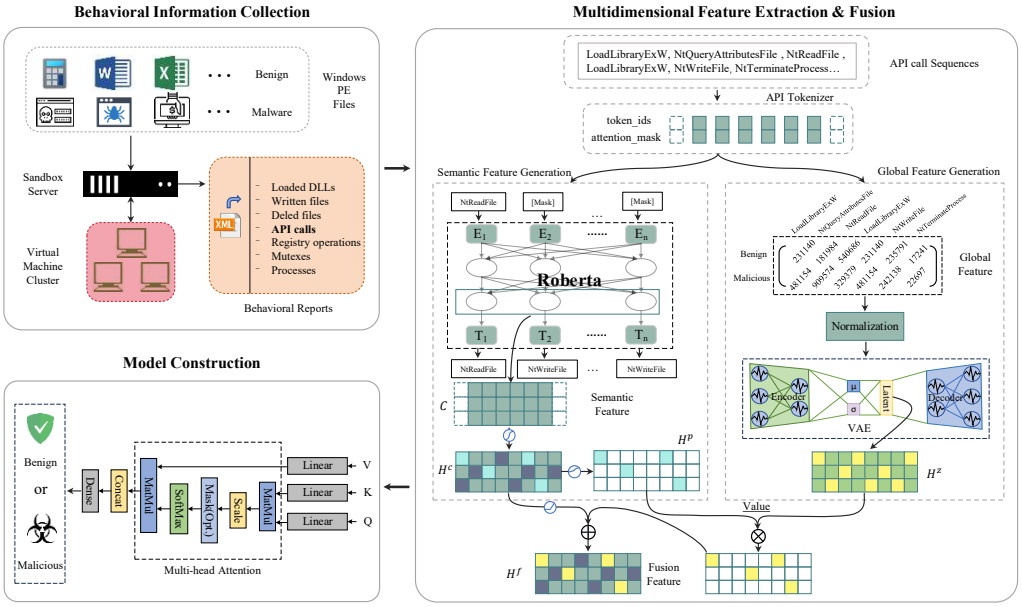


Figure 1: The framework of the proposed malware detection system consists of three main components: behavior information collection, multidimensional feature extraction and fusion, and model construction.

ity to generalize detection of new and variant malware.

3. Methodology

3.1. Overview

The framework of the malware detection system proposed in this paper is illustrated in Fig. 1, consisting primarily of three components: behavior information collection, multidimensional feature extraction and fusion, and model construction. Behavior Information Collection: The behavior information collection module obtains dynamic behavior reports containing API sequences and their parameters by uploading executable files to a sandbox. Multidimensional Feature Extraction and Fusion: The multidimensional feature extraction module extracts semantic and global features from dynamic reports. The feature fusion module generates multidimensional fusion features of API calls' context semantics and global information through a gating mechanism, enhancing the semantic information of API sequence features. Model Construction: The model construction module adopts a multi-head

attention mechanism to build a detection model based on enhanced semantic API sequence features, thereby improving the overall detection performance of the proposed solution.

3.2. Behavior Information Collection and Preprocessing

Dynamic behavior reports are generated by executing the program under analysis within a controlled sandbox environment, where system-level behaviors produced during execution are monitored and recorded. Among these behaviors, API call sequences directly reflect the interactions between the program and the operating system, as well as the underlying logic of functional implementation. However, raw dynamic behavior reports typically contain substantial redundancy and noise, such as consecutively repeated API calls caused by loop structures or state polling mechanisms. Such redundant behaviors not only increase sequence length but may also obscure critical behavior patterns, thereby degrading the model's ability to effectively capture and represent malicious behaviors.

Therefore, we preprocess the dynamic behavior reports using Algorithm 3.1 to remove redundant information and effectively extract API sequence features. The algorithm extracts API sequences from dynamic reports and eliminates consecutive duplicate API calls to ensure both the compactness of the sequence and the accuracy of the extracted information. For instance, the sequence {a, a, a, b, c, c, d, a, . . .} is transformed into {a, b, c, d, a, . . .}. During this process, we use whitespace as a delimiter between each API and standardize the sequence length to a maximum of 512 tokens for RoBERTa, thereby maximizing the retention of sequence information. Additionally, considering that the tokenization of API sequences by the RoBERTa model may compromise the integrity of the call sequence, and the relatively low number of actual API calls in dynamic analysis reports may lead to model overfitting, we construct a dictionary using publicly available APIs[1] from Cuckoo, Microsoft's publicly available Windows system APIs [2], and APIs contained in the training set to convert API sequences into vector representations.

3.3. Multidimensional feature extraction and fusion

The objective of the multidimensional feature extraction and fusion module is to dynamically generate multidimensional fusion features based on

[1] `https://github.com/cuckoosandbox/cuckoo/wiki`

[2] `https://learn.microsoft.com/windows-hardware/drivers/ddi/_kernel`

Algorithm 3.1: ExtractAPISequence

Input : Dynamic behavior reports
Output: Formatted API sequence S
Initialization:
len ← fixed length; prev ← empty; dict ← API dictionary;
foreach *line in file* **do**
  API ← `extract`(line);
  API ← `lowercase`(API);
  if $API \in dict$ ***and*** $API \neq prev$ **then**
    append dict[API] to S separated by space;
  end
  prev ← API;
end
if $|S| > len$ **then**
  $S \leftarrow S[1{:}\text{len}]$;
end
else if $|S| < len$ **then**
  pad S to length len;
end
prepend [cls] to S; append [sep] to S;
return S;

contextual semantics and global information, thereby enhancing the semantic information of the API sequences obtained from the behavior information collection module through the gating mechanism. The semantic feature generation module utilizes the RoBERTa model to explore and describe the contextual semantic information, accurately describing the semantic correlations between API call sequences. Meanwhile, the global feature generation module samples and encodes API call frequencies using a variational autoencoder to create a universal representation of global API call features. This representation captures the behavioral patterns of API calls within the system. The multidimensional weight control module, utilizing the gating mechanism, effectively adjusts the weights of various modal features to ensure their optimal interaction, thereby generating semantically enhanced API sequence features.

3.3.1. Semantic feature generation

To address the issue of insufficient semantic feature extraction from API sequences, we employ the RoBERTa model to extract semantic features from API call sequences with temporal characteristics and map them to an information space, as illustrated in Fig. 2. This model treats the API call sequences of each malicious software as a whole, avoiding the loss of information and reducing computational efficiency that can occur when splitting API call sequences into multiple sentences. This approach aids in enhancing the performance of the model in malicious software detection and classification.

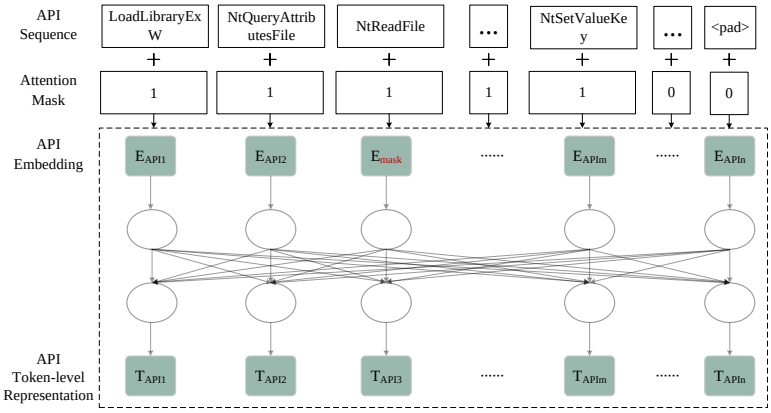


Figure 2: The semantic feature generation module based on RoBERTa.

We first standardize the length of the API sequence to the maximum input length of 512 for RoBERTa, resulting in a sequence $s = \{LoadLibraryExW, NtQueryAttributesFile, NtReadFile, \ldots, NtSetValueKey, \ldots, < pad >\}$. We use attention mask tokens to mark the padding symbols, reducing computational costs by ensuring the model focuses only on the relevant parts of the sequence. Subsequently, we train the API sequences using the dynamic masking mechanism outlined in Eq. 1 to accurately obtain the semantic feature mapping graph $C = \{T_{API1}, T_{API2}, T_{API3}, \ldots, T_{APIm}, \ldots, T_{APIn}\}$.

$$C = RoBERTa(s) \tag{1}$$

Next, the semantic feature graph C is mapped to the information space through dense layers:

$$H^c = \sigma(W^c C + b^c) \tag{2}$$

where $\sigma(\cdot)$ denotes the sigmoid function. Eq. 2 is utilized to map the semantic feature graph C to the representation of semantic features of the API sequence H^c.

3.3.2. Global feature generation

In this study, we employed a Variational Autoencoder (VAE) to map discrete API statistical vectors into a continuous space, aiming to capture a global representation of API statistics from dynamic analysis reports of both malicious and benign software. This approach facilitates the exploration of malicious behavior patterns in dynamic behavior reports. First, we extract the global frequency features of API calls from the training set. This involves computing the occurrences of individual APIs in both malicious and benign programs and then normalizing these occurrences by the total count. Next, we represent each software's API call sequences as frequency matrices, which are then input into the VAE for encoding. This step maps the API frequency features of executable programs into vector representations in the latent space, facilitating the extraction and analysis of global API call features.

Let X denote the statistical vectors generated for the API call sequences of all executable programs in the training set, represented as $X = \{x^{(i)}\}_{(i=1)}^{N}$, where x represents the statistical vector of different program API sequences, with a total count of N. The Variational Autoencoder (VAE) generates the probability distribution $q_\phi(z \mid x^{(i)})$ of latent variables $z^{(i)}$ for the input statistical vector $x^{(i)}$ through the encoder, where ϕ represents the set of encoder parameters containing all weights and biases learned during the encoding process. The goal of the encoding process is to map the API global features $x^{(i)}$ to the probability distribution $q_\phi(z \mid x^{(i)})$ in the latent space. Subsequently, the reparameterization trick is applied to sample the latent variables $z^{(i)}$, enabling the characterization of global program behaviors at the API-call level.

The goal of the decoding process is to map the sampled latent variable values $z^{(i)}$ back to the input space through the decoder for backpropagation, resulting in the reconstructed probability distribution $p_\theta(x^{(i)} \mid z^{(i)})$. The loss function, as shown in Eq. 3, primarily consists of two parts: the expectation term $\mathbb{E}$ measures the generative capability of the model, while the KL term ensures that the learned latent API representations are close to the prior distribution.By jointly minimizing the two loss terms, the latent global API representation z is encouraged to approximate the standard Gaussian

prior distribution $p(z) = \mathcal{N}(0, I)$. This process enables the encoder to generate latent vectors that effectively capture global API-level features, while preventing excessive dispersion or degeneration of the latent space.

$$\begin{aligned}\mathcal{L}\left(\theta, \phi; x^{(i)}\right) = &-\mathbb{E}_{q_\phi\left(z|x^{(i)}\right)}\left[\log p_\theta\left(x^{(i)} \mid z\right)\right] \\ &+\mathrm{KL}\left(q_\phi\left(z \mid x^{(i)}\right) \| p(z)\right),\end{aligned} \tag{3}$$

To enable efficient optimization, we employ the reparameterization trick shown in Eq. (4), which allows the model to better fit the global feature distribution of API sequences. For the i-th sample, the latent variable $z^{(i)}$ is formulated as:

$$z^{(i)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \tag{4}$$

where $\mu^{(i)}$ and $\sigma^{(i)}$ denote the mean and standard deviation of the latent distribution output by the encoder, respectively, $\odot$ represents element-wise multiplication, and ϵ is random noise sampled from a standard normal distribution. Consequently, the approximate posterior distribution modeled by the encoder can be expressed as:

$$q_\phi\left(z \mid x^{(i)}\right) = \mathcal{N}\left(z; \mu^{(i)}, \sigma^{(i)2} I\right), \tag{5}$$

where μ denotes the distribution mean, σ represents the standard deviation, and $\sigma^2 I$ is the covariance matrix of the multivariate Gaussian distribution characterizing the global API features.

Finally, the latent vector representation z of the global information of executable programs, obtained through the VAE encoder, is projected into an information space of equal dimensionality as the semantic features, yielding the vector representation H^z of the global features.

3.3.3. Multidimensional feature weight control

As shown in Fig. 3, we apply the *Gate* function of the gating mechanism [31] to fuse the semantic information H^c and the global information H^z in the design of the multidimensional feature weight control module. This module dynamically adjusts the feature weights between modalities, leveraging the global features to obtain enhanced semantic API sequence features H^f.

$$H^f = Gate\left(H^c, H^p, H^z, \varepsilon\right) = Relu\left(H^c\right) + Valve\left(H^p, \varepsilon\right) \odot H^z \tag{6}$$

In Eq. 6, $Relu(\cdot)$ represents the activation function, and $\odot$ denotes element-wise multiplication. H^p is the per-API probability representation

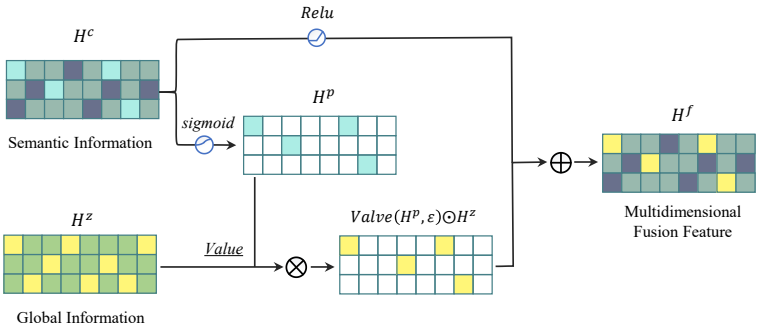


Figure 3: Multidimensional feature weight control module based on gating mechanism.

vector obtained by mapping the semantic feature H^c through a linear layer, where each dimension corresponds to the discriminative probability of a specific API call. The $Valve$ function aims to remove API calls from the global information H^z that have low confidence in classification. Concretely, for every API $a \in H^p$,

$$Valve\,(a,\varepsilon) = \begin{cases} a, & \text{if } 0.5-\varepsilon \le a \le 0.5+\varepsilon \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

where the parameter ε represents the confidence threshold, which is used to adjust the degree of fusion of the global information H^z. Specifically, if $\varepsilon = 0$, all global information will be discarded. whereas if $\varepsilon = 0.5$, all global information will be accepted.

Therefore, this paper utilizes the $Valve$ function in the gating mechanism to dynamically adjust the global feature information. Subsequently, the *Gate* function integrates the global and semantic information, resulting in the multidimensional fused features H^f of the API sequences.

3.4. Model Construction

This paper employs a multi-head attention mechanism to handle the generated multidimensional fused features H^f, enabling each attention head to capture contextual relational information of enhanced semantic API sequence features from different perspectives, thereby enhancing the detection performance of the model. As illustrated in Fig. 4, the process begins by generating the initial queries, keys, and values from the multidimensional fused features H^f, where $H^f = Q = K = V$. Next, as depicted in Eq. 8, different feature

heads $\{W^Q, W^K, W^V\}$ are generated through Linear Layer transformations:

$$\begin{aligned} \text{multihead} = &\left\{\left\{W_0^Q, W_0^K, W_0^V\right\},\right. \\ &\left.\left\{W_1^Q, W_1^K, W_1^V\right\}, \ldots, \left\{W_h^Q, W_h^K, W_h^V\right\}\right\} \end{aligned} \tag{8}$$

Subsequently, by utilizing Eq. 9, the matrices Q, K, and V are multiplied with each distinct feature head, resulting in varying attention weights for the API sequence features:

$$Q_i = QW_i^Q, K_i = KW_i^K, V_i = VW_i^v \tag{9}$$

Then, the attention results for each head are computed separately using Equation 10:

$$\text{Attention} = softmax\left(\frac{Q_i K_i^T}{\sqrt{K_i}}\right) V_i \tag{10}$$

Thereafter, all attention results from the different heads are concatenated and multiplied by the weight matrix W_0 to obtain the enhanced semantic feature representation of the API sequence, as defined in Equation 11:

$$\begin{aligned} \text{MultiHead}\,(Q, K, V) = \text{Concat}\left(softmax\left(\frac{Q_0 K_0^T}{\sqrt{K_0}}\right) V_0,\right. \\ \left. softmax\left(\frac{Q_1 K_1^T}{\sqrt{K_1}}\right) V_1, \ldots, softmax\left(\frac{Q_h K_h^T}{\sqrt{K_h}}\right) V_h\right) W_0 \end{aligned} \tag{11}$$

where W_0 is the weight matrix used to integrate the outputs of multiple attention heads to combine the attention results.

Finally, the enhanced semantic API sequence features outputted by the multi-head attention mechanism are dimensionally reduced through a pooling layer before being fed into a fully connected neural network (FCNN) to construct the detection model for malicious software.

4. Experiments design and evaluation

4.1. Datasets

We employed the dynamic dataset from Datacon [32] to validate the effectiveness of our proposed approach. Datacon dynamic dataset utilizes sandbox technology to monitor the behavior of executable programs, providing

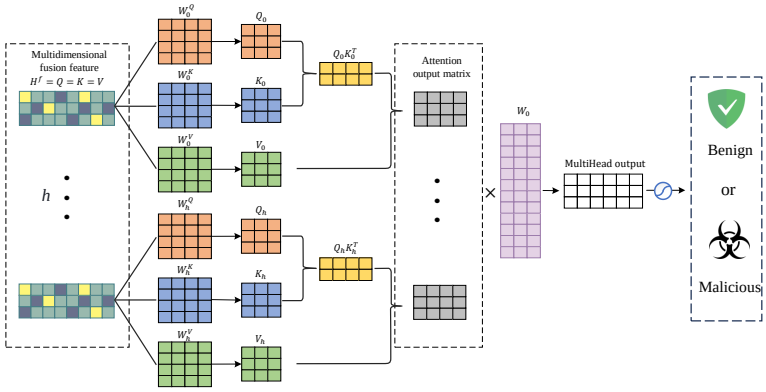


Figure 4: Classification model.

detailed records of file system operations, network communications, and registry modifications. This dataset includes a total of 49,207 dynamic behavior reports from malicious samples and 20,000 from benign samples. To ensure the balance of different types of samples in our experiments and to align with the dataset partitioning methods used in most existing research, we utilized two datasets to evaluate the proposed method, as presented in Table 1: TrainDataset, which consists of 20,000 execution traces of Windows PE files, with 10,000 benign and 10,000 malicious files, and TestDataset, which has a similar structure but with slight differences in the number of tracked APIs, containing 94 APIs in TrainDataset and 98 APIs in TestDataset.

To assess the generalization capability of our proposed approach, we also employed the publicly available datasets used in the study by Amer et al. [25]: Ki [33] and Catak [34] datasets. Unlike the Datacon dataset, which uses a self-developed sandbox, the Ki and Catak datasets utilize the publicly available Cuckoo sandbox, capturing a total of 321 different API calls. The Ki dataset comprises dynamic API call sequences from 23,146 malicious samples and 21,116 benign samples, while the Catak dataset contains dynamic API call sequences from 7,107 malicious samples and 169 benign samples.

Furthermore, to evaluate the stability of the proposed method and to expand its coverage for multi-class detection, this study also incorporates the AliYun dataset [35], which contains API call sequences for eight types of software: Normal (4978), Ransomware (502), Miner (1196), Distributed Denial-of-Service (DDoS) (820), Worm (100), Virus (4289), Backdoor (515), and Trojan (1487).

Table 1: Datasets Used in the Paper

Dataset	Task Type	Malware	Benign	Classes	Evaluation Protocol	API Types
Datacon (TrainDataset)	Binary	10,000	10,000	2	5-fold CV (training & validation)	94
Datacon (TestDataset)	Binary	10,000	10,000	2	Testing	98
Ki	Binary	23,146	21,116	2	5-fold CV	321
Catak	Binary	7,107	169	2	5-fold CV	321
AliYun	Multi-class	8,009	4,978	8	5-fold CV	239

4.2. Experimental setup

To validate and evaluate the effectiveness and generalization ability of the proposed malware detection scheme, a series of experiments were conducted on multiple datasets. As shown in Table 1, this section employs a five-fold cross-validation method on the TrainDataset and TestDataset based on the Datacon [32] dataset for validation and testing. Additionally, the generalization capability of the scheme is assessed on the Ki [33] and Catak [34] datasets. In addition, to further validate the detection performance of this scheme under imbalanced data distribution, we partition the Datacon [32] dataset into different distributions with a 7:3 ratio for training and testing.

Furthermore, five-fold cross-validation is conducted on the multi-class AliYun dataset [35] to evaluate the model's adaptability and generalization capability when confronted with long-tail distributions and complex malware categories.

This paper assesses the detection performance of the system using five performance metrics commonly utilized by malware researchers: accuracy, precision, recall, F1 score, and false positive rate (FPR) (see Eqs. 12 - 16).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{15}$$

$$FPR = \frac{FP}{FP + TN} \tag{16}$$

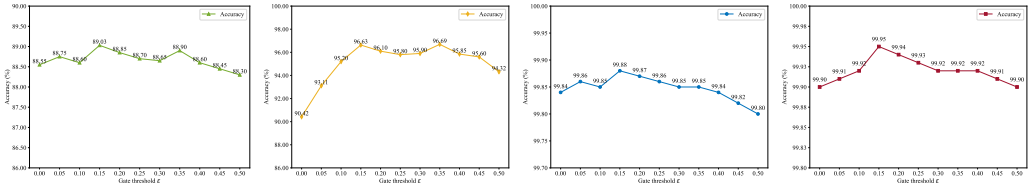
where TP (True Positives) is the number of malicious software samples correctly detected, FP (False Positives) is the number of benign executables predicted as malicious software, FN (False Negatives) is the number of malicious software detected as benign executables, and TN (True Negatives) is the number of benign samples correctly detected.

4.3. Experiment Results and Analysis

In section 4.3.1, to further investigate the effectiveness of the proposed approach, we adjust the gate threshold on the Datacon dataset to obtain the weight between global and semantic features that enhances detection performance. Subsequently, in section 4.3.2, we conduct ablation experiments based on the Datacon dataset to validate the effectiveness of the proposed semantic feature generation module and multidimensional weight control module. Then, in section 4.3.3, we utilize the multi-head attention mechanism further to address the temporal relationships in feature multidimensional fusion, aiming to ascertain its efficacy in enhancing the performance of the detection model. Finally, in section 4.3.4, experiments on the Ki and Catak datasets are conducted to validate the robustness and generalization ability of the proposed detection scheme.

4.3.1. Impact of Different Valve Thresholds on the Model

To assess the dynamic adjustment capability of the weight control module constructed based on the gating mechanism, five-fold cross-validation was performed on the Datacon validation dataset. By tuning the threshold ε, we aimed to achieve an optimal weight allocation between global and semantic features that could effectively enhance detection performance. The results of incorporating the gating mechanism and adjusting the threshold ε in the BiLSTM, Text-CNN, BERT, and RoBERTa models are shown in Fig. 5. At $\varepsilon = 0$, the detection model solely relies on semantic information from API sequences, while at $\varepsilon = 0.5$, the models fully adopt the enhanced semantic API sequence features generated from both global and semantic information. Although this increases feature diversity, it also introduces a substantial amount of redundant information from the global features, leading to a decrease in detection performance. Notably, at $\varepsilon = 0.15$, the multidimensional features integrated by RoBERTa with the gating mechanism effectively extract the global features of the API sequences while eliminating noise. Moreover, across different base models (BiLSTM, Text-CNN, BERT,



(a) (b) (c) (d)

Figure 5: Effect of the gating mechanism on the model performance with different thresholds ε. (a) BiLSTM with Gate. (b) Text-CNN with Gate. (c) BERT with Gate. (d) RoBERTa with Gate

and RoBERTa), the model performance exhibits a consistent trend with respect to ε: the performance gradually improves within a small threshold range and starts to decline once the threshold exceeds a certain value. This indicates that the gating mechanism is not highly sensitive to the threshold parameter and maintains stable performance within a reasonable range. Using this approach, a classification accuracy of 99.95% was achieved on the Datacon TrainDataset, surpassing detection schemes based on BiLSTM, Text-CNN, and BERT models. These results demonstrate that the enhanced semantic API sequence features generated through the adjustment of multi-dimensional features by the gating mechanism can improve the performance and robustness of the detection model.

4.3.2. Evaluating the Impact of Various Components on Outcomes

This section aims to validate the effectiveness of two proposed modules: the semantic feature generation module based on RoBERTa and the multidimensional weight control module based on the gating mechanism. Comparative analysis with existing baseline models, including BiLSTM [11], Text-CNN [10], and BERT [21], will be conducted to verify the effectiveness of utilizing the RoBERTa model for semantic feature extraction. Furthermore, to assess the effectiveness of the gating mechanism, we will compare the detection models employing the gating mechanism with those that do not.

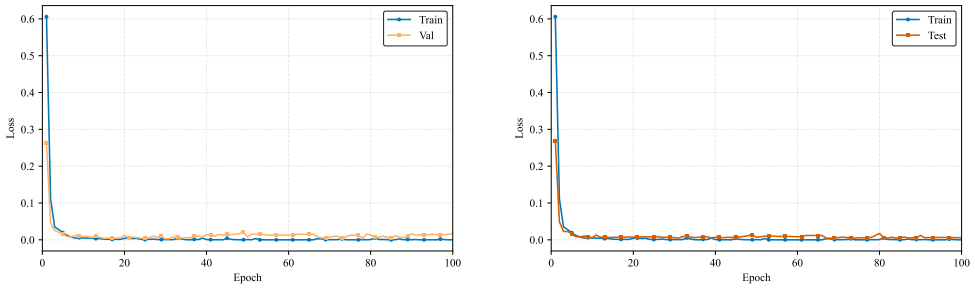
The experimental results in Table 2 indicate that the detection models constructed using BiLSTM, Text-CNN, BERT, and RoBERTa achieve accuracies exceeding 84%, demonstrating the effectiveness of mining API sequences for detecting malware. RoBERTa outperforms BiLSTM and Text-CNN models with accuracy improvements of 13.87% and 4.37%, respectively, during training. This suggests that RoBERTa can more accurately capture

the overall semantics and logical relationships of API sequences, thereby providing a more comprehensive understanding of executable program behavior. In contrast, although BiLSTM can capture long-term dependencies in API sequence data, its bidirectional structure to some extent limits its understanding of global information within API sequences. Text-CNN, on the other hand, processes sequence features at different scales through convolutional operations, enabling it to capture some local patterns in sequences. However, its understanding of the overall semantics of API sequences in dynamic behavioral reports is relatively weak. Compared to the BERT model, RoBERTa eliminates the next sentence prediction task, thereby avoiding the issues of information loss and decreased computational efficiency resulting from sequence segmentation. The model's dynamic masking approach also allows for a better understanding of API contextual information, effectively capturing semantic features within API sequences. As a result, RoBERTa achieves classification accuracies of 99.92% and 99.72% during the training and testing phases, respectively.

Comparing the detection performance before and after incorporating the gating mechanism validates its effectiveness in handling multidimensional features of API sequences. During testing, the four sequence feature processing schemes constructed using the gating mechanism exhibited improvements in classification accuracy of 3.93%, 1.7%, 0.27%, and 0.18%, respectively, compared to the schemes without the gating mechanism. Particularly, the RoBERTa model based on the gating mechanism achieved classification accuracies of 99.95% and 99.90% during training and testing, respectively. This indicates that the enhanced semantic API sequence features generated using the gating mechanism can effectively enhance the detection performance of sequence processing models. Furthermore, by comparing the approach that directly incorporates frequency-based features with the gating mechanism-based method, it can be observed that although RoBERTa is able to partially compensate for model capacity during training, achieving relatively satisfactory performance under five-fold cross-validation, relying solely on frequency information introduces additional noise when faced with unseen data, particularly when the test set contains unknown APIs, leading to a decline in overall model performance. In contrast, by introducing the gating mechanism, the model can selectively filter multi-dimensional features based on contextual semantic information, effectively suppressing irrelevant or noisy features and achieving more stable and consistent performance across both training and test sets.

Table 2: Performance comparison with baseline models

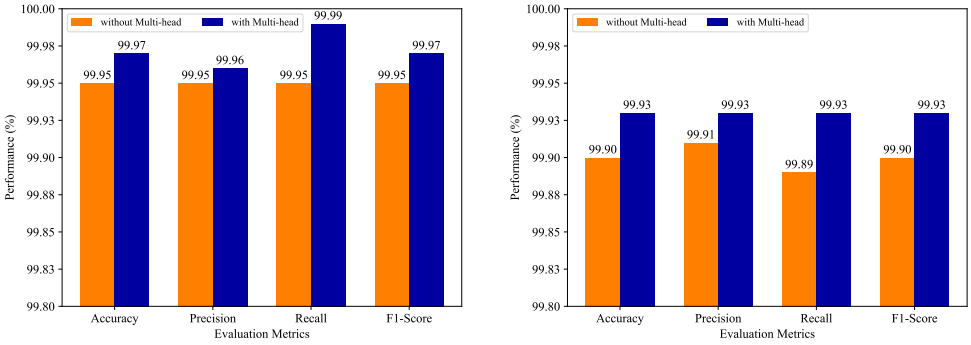
Method	Validation Results on TrainDataset (%)				Test Results on TestDataset (%)			
	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
BiLSTM	86.10	80.88	94.55	87.18	84.95	79.20	94.80	86.30
BiLSTM+Gate	89.03	86.25	92.92	89.44	88.88	86.22	92.58	89.27
Text-CNN	95.60	95.75	95.44	95.59	94.77	95.53	93.95	94.73
Text-CNN+Gate	96.63	96.75	96.50	96.62	96.47	96.28	96.67	96.47
BERT	99.83	99.80	99.85	99.83	99.58	99.35	99.80	99.58
BERT+Gate	99.88	99.87	99.89	99.88	99.85	99.85	99.85	99.85
RoBERTa	99.92	99.95	99.90	99.92	99.72	99.60	99.85	99.73
RoBERTa+Frequency	99.85	99.90	99.95	99.95	98.90	98.34	98.45	98.90
RoBERTa+Gate	**99.95**	**99.95**	**99.95**	**99.95**	**99.90**	**99.91**	**99.89**	**99.90**



(a) (b)

Figure 6: Overfitting diagnostics for one fold of the 5-fold cross-validation (RoBERTa+Gate). (a) Training and validation loss curves. (b) Training and test loss curves.

In addition, the RoBERTa model with a gating mechanism demonstrated strong stability during training. To systematically evaluate the model's training dynamics, we conducted 100 training rounds, performing testing after each training and validation phase to observe the model's performance trends throughout the entire training process. As shown in Fig. 6, the model achieves a rapid and synchronized decrease of both training and validation losses within approximately 10 epochs, followed by stable convergence. The training and testing loss curves exhibit a high degree of consistency, with no evident overfitting. Combined with the Early Stopping strategy (monitor = train loss, patience = 5), these results indicate that the gating mechanism not only enhances the final detection performance but also ensures stable training and robust generalization capability.



(a) (b)

Figure 7: Performance Comparison with and without Multi-head Attention. (a) Performance on Datacon TrainDataset. (b) Performance on Datacon TestDataset.

4.3.3. Effectiveness of the Model Classifier

Due to the enhanced semantic API sequence features generated by integrating semantic and global features with the gating mechanism still maintaining a temporal structure distribution, relying solely on fully connected neural networks to process these features might overlook their temporal information, potentially leading to model overfitting. Hence, this paper employs a multi-head attention mechanism to capture associated features from different perspectives within the enhanced semantic API sequence features, effectively handling the temporal relationships within the fused features, and thereby enhancing the model's ability to detect and generalize malicious software. As shown in Fig. 7, the detection model constructed using the multi-head attention mechanism achieved an increase in classification accuracy from 99.95% to 99.97% during training and from 99.90% to 99.93% during testing compared to the baseline model using fully connected neural networks. Therefore, the adaptive adjustment of attention weights for each head through the multi-head attention mechanism can suppress noise within the enhanced semantic API sequence features, enhance the model's understanding of semantic and structural information within sequence features, and effectively improve detection performance.

4.3.4. Performances in the Ki and Catak dataset

We conducted generalization experiments using the datasets proposed by Ki et al. [33] and Catak F et al. [34] to validate the robustness and

Table 3: Performances Comparison with current work in the Ki and Catak dataset

Method	Validation Results on Ki Dataset (%)				Validation Results on Catak Dataset (%)			
	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
Amer et al. [25]](2020)	99.90	99.80	99.90	99.90	98.00	99.40	98.70	98.70
Xu et al. [21](2021)	99.98	100	99.98	99.99	99.82	99.91	99.91	99.91
Proposed Model	**100.00**	**100.00**	**100.00**	**100.00**	**99.96**	**99.96**	**100.00**	**99.98**

generalization ability of the proposed approach across different datasets. All experiments were conducted using five-fold cross-validation to ensure the reliability and robustness of the results, as presented in Table 3.

Amer et al. [25] constructed a malware detection system by capturing the global structural information of API call sequences. In contrast to the aforementioned studies, the method proposed in this paper, which utilizes a gating mechanism to construct enhanced semantic API sequence features, enhances the sensitivity of the model to detect different malware samples. It achieved an increase of 0.1% and 1.96% in detection accuracy on the Ki dataset and the Catak dataset, respectively. Compared to the method of only fine-tuning with the Bert model used by Xu et al. [21], our approach, which employs the RoBERTa model to extract semantic information between API call sequences, enhances the model's ability to capture potential relationships between API calls. It achieved classification accuracies of 100.00% and 99.96% on the Ki dataset and Catak dataset, respectively. These results demonstrate the proposed method's robust generalization and adaptability, rendering it suitable for various malware detection tasks.

4.4. Comparison with Existing Work

In this section, we conduct a comprehensive comparison and analysis of our proposed malware detection approach against relevant studies using the Datacon dataset 4.4.1 4.4.2. By evaluating the technical methodologies, research perspectives, and experimental outcomes, we highlight the performance, advantages, and distinctions of our approach within the field of malware detection. Additionally, in Section 4.4.3, we further evaluate the adaptability of the proposed approach through experiments on the AliYun dataset and assess the model's average detection latency to comprehensively understand its practical performance. Finally, we present a critical discussion on the limitations of our method and potential areas for improvement in 5.

4.4.1. Comparison with Current Work in Balanced Distribution

Table 4 presents the results of related studies conducted on the Datacon dataset, utilizing the same data volume (40K) and data distribution (20K malicious: 20K benign). Ndibanje et al. [28] utilized the k-nearest neighbor algorithm to identify and classify malware by computing the similarity between API sequences and MSDN APIs, achieving a classification accuracy of 85.54% in the testing phase. Yang et al. [36] integrated multiple machine learning algorithms to process API sequences and their parameters, constructing a classification system with a detection rate of 99.67% on the training set. While machine learning-based methods have demonstrated some effectiveness in malware detection tasks, their reliance on feature engineering, limited generalization capabilities, and insufficient representation power for time-series data constrain their reliability and effectiveness when facing complex malware and unknown malicious behavioral patterns. In contrast to the machine learning-based detection methods mentioned earlier, the approach constructed in this paper leverages deep learning techniques, enabling automatic learning of high-level feature representations from API sequences without manual intervention in feature design and selection. Moreover, deep learning methods excel in capturing complex structures and patterns in the data by learning abstract representations of API sequences. This capability enhances the performance and generalization capability of the model. The achieved classification accuracy of 99.93% on the test set validates the effectiveness of our proposed solution and demonstrates its robust generalization ability to unseen data.

Amer et al. [25] employed the Markov chain algorithm to extract global structural information from API call sequences for constructing a malware detection system, achieving a detection accuracy of 88.63% during the training phase. Additionally, Zhang et al. [11] utilized Gated-CNNs and LSTM to explore API and its parameter features, achieving classification accuracies of 97.97% and 91.76% respectively, during the training and testing phases. Building upon the work of Zhang et al. [11], Li et al. [12] applied convolutional methods and LSTM models to mine API sequences and their intrinsic features, achieving detection accuracies of 99.71% and 99.67% on the Datacon dataset. In contrast, by employing RoBERTa, we further explore the potential semantic features within API sequences, enabling a more accurate grasp of the overall semantic and logical relationships within the sequences and a more comprehensive understanding of executable program behavior

Table 4: Performances Comparison with current work in the Datacon dataset

Method	Parameters	Validation Results on TrainDataset (%)				Test Results on TestDataset (%)			
		Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
Ndibanje et al. [28](2019)	No	96.40	96.56	96.21	96.40	85.54	94.98	75.04	84.84
Yang et al. [36](2019)	Yes	99.67	-	99.67	99.67	-	-	-	-
Amer et al. [25](2020)	No	88.63	85.27	88.81	86.92	67.11	61.62	90.70	73.38
Zhang et al. [11](2020)	Yes	97.97	98.69	97.26	97.97	91.76	97.64	85.59	91.22
Chen et al. [37](2022)	Yes	98.09	98.48	97.69	98.08	96.78	96.82	96.78	96.80
Ding et al. [38](2022)	Yes	97.82	97.84	97.83	97.82	-	-	-	-
Chen et al. [10](2022)	Yes	98.16	98.66	97.66	98.16	98.52	98.63	98.41	98.52
Li et al. [12](2022)	No	99.71	99.63	99.78	99.70	99.67	99.64	99.73	99.69
Du et al. [39](2023)	Yes	98.21	98.70	98.71	98.21	97.63	97.64	97.63	97.64
Chen et al. [40](2024)	Yes	99.50	99.40	99.80	99.60	98.30	97.80	97.00	98.40
Hou et al. [41](2025)	Yes	98.59	98.37	98.82	98.78	98.40	98.24	98.55	98.62
Proposed model	**No**	**99.97**	**99.96**	**99.99**	**99.97**	**99.93**	**99.93**	**99.93**	**99.93**

patterns. This approach, to some extent, addresses the issues of loss of sequence context information in Markov chain methods and the difficulty of LSTM methods in capturing long-range dependencies and handling complex semantics.

In recent research, researchers have sought to reveal the underlying behavioral patterns of executable programs by analyzing API call sequences and parameters. In 2022, Chen et al. [37] utilized a weighted approach of logistic regression to evaluate APIs and their parameters, employing an API call process graph for malware detection, achieving classification accuracies of 98.09% and 96.78%, respectively. In the same year, Ding et al. [38]] analyzed API call relationships between different processes using API parameters, proposing a detection scheme based on graph convolutional neural networks that achieved a classification accuracy of 97.82%. Subsequently, Du et al. [39] employed logistic regression and machine learning methods to weight and score behaviors in API calls and parameters, constructing a detection model that achieved classification accuracies of 98.21% and 97.63% during training and testing phases, respectively. In addition, Hou et al. [41] constructed a parameter encoder to map different types of parameters into feature vectors, and further applied a clustering-based discretization to these parameter features to enhance the expressiveness of API representations, achieving an accuracy of 98.59% and an F1-score of 98.40%. However, while malware detection systems constructed using APIs and their parameters offer more refined feature representation capabilities, the complexity of feature selection and parameter tuning may result in increased time overhead and reduced system generality and ability to identify unknown malicious behaviors. In contrast, our proposed solution focuses on extracting semantic and global information from API call sequences, simplifying the complexity of feature

processing, and enhancing system efficiency and generality. Meanwhile, our approach eliminates reliance on parameters, thereby improving the system's adaptability to unknown malicious behavior patterns.

Some researchers employ domain knowledge-based methods for dynamic malware detection. For example, Chen et al. [10] conducted preliminary clustering analysis on APIs using domain knowledge such as API operation permissions and file names in call paths, along with statistical rules. They further integrated Text-CNN models for malware detection. Chen et al. [40] utilized threat intelligence from Cyber Threat Intelligence (CTI) to label the security sensitivity levels of APIs, integrating this information with the original API sequences into a feature space for detection using deep learning models. While these approaches prove effective in dynamic malware detection tasks, relying on domain knowledge for malware detection is subject to expert experience and subjective understanding, which may not promptly adapt to changes in malware behavior and could potentially introduce bias or misjudgment into detection systems. Additionally, acquiring and updating domain knowledge for large-scale malware detection tasks entails significant time and cost. In contrast, this paper leverages the RoBERTa model, eliminating the need for manual analysis and feature engineering, and enabling adaptive extraction of semantic information from API sequences to comprehensively capture complex semantic relationships between features. Furthermore, the enhanced semantic API sequence features generated through the gating mechanism enhance the model's sensitivity to detecting various malware samples, thereby accommodating different detection requirements across diverse scenarios.

In summary, this paper employs the gating mechanism to dynamically adjust the global information extracted by the variational autoencoder to enhance the semantic information captured between API call sequences by the RoBERTa model. By constructing a classification model using multi-head attention mechanisms, high-performance detection of malicious software is achieved. The classification accuracy of 99.93% and an F1 score of 99.93% achieved on the Datacon dataset demonstrate the robust performance of the proposed approach in malware detection tasks, providing valuable insights and references for related research and applications.

4.4.2. Comparison of Different Data Distributions

To simulate the real-world scenario of malware being hidden among a large volume of benign software, we evaluated the performance of the pro-

posed scheme under different data distributions. The experimental results, as shown in Figure 6, indicate that the accuracy of the scheme remains stable within the range of 99.93% to 99.97% across various data distributions. Although there is a slight decrease in the F1 score, it still maintains above 99.60%. Notably, when the ratio of malicious to benign samples decreases to 1K:20K, the F1 score still reaches 99.68%, demonstrating the robustness of the proposed scheme in handling imbalanced data. Furthermore, when the ratio of malicious to benign samples is 16K:20K, 12K:20K, and 8K:20K, the false positive rate remains below 0.03%, further confirming that our scheme effectively reduces false positives and mitigates the potential threat of malware to endpoint systems.

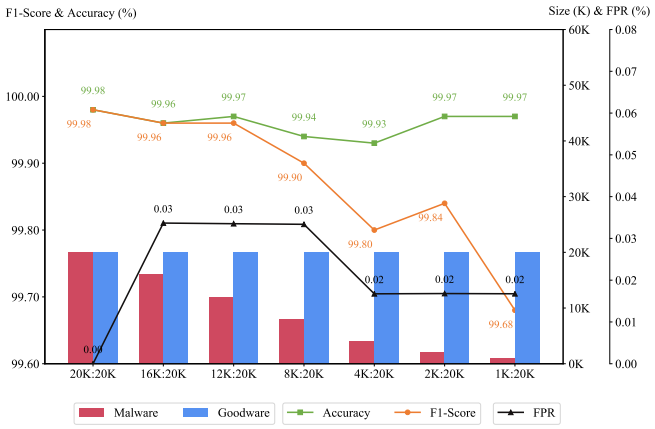


Figure 8: Comparison of F1-score, Accuracy and False Positive Rate (FPR) for Different Data Distributions.

To evaluate the malware detection performance of the proposed scheme under different data distributions, we analyzed the model's true positive rate (TPR) and false positive rate (FPR) at various decision thresholds through ROC curve analysis, providing a visual representation of the scheme's performance. We also quantified the area under the ROC curve (AUC) to assess the detection capability and robustness of the scheme. The experimental results, as shown in Fig. 9, indicate that the AUC value consistently remains above 99.82, regardless of variations in the ratio of malicious to benign samples. This demonstrates that the proposed scheme exhibits strong robustness and detection performance in both balanced and imbalanced data distributions, as well as in simulated real-world environments.

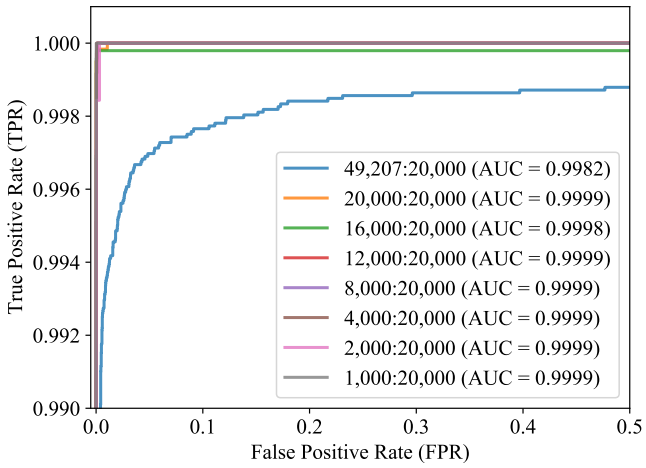


Figure 9: ROC Curves and AUC Values on Balanced and Imbalanced Datasets.

4.4.3. Comparison with Current Work in the AliYun dataset

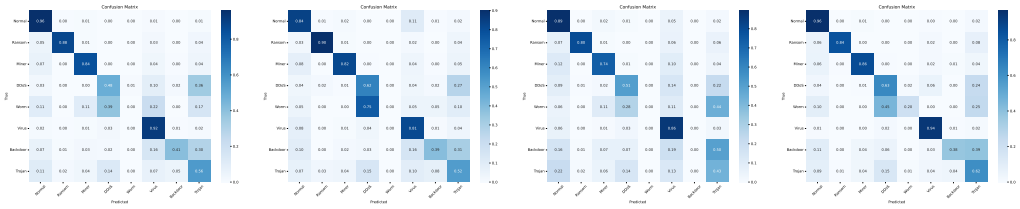
To further evaluate the stability and generalization capability of the proposed method in multi-class scenarios, this section incorporates the AliYun dataset [35] and reproduces relevant methods using five-fold cross-validation to ensure comparability of experimental results, as shown in Table 5. Li et al.[13] constructed semantic representations based on API call sequences using a CNN+RNN-based framework, achieving 83.84% multi-class accuracy and 95.39% binary accuracy on the AliYun dataset with relatively fast detection speed. However, due to the inherent limitations of recurrent structures in modeling long-range dependencies, the CNN+RNN model still exhibits bottlenecks in distinguishing highly similar classes and capturing complex semantic patterns. In contrast, Transformer architectures excel in handling long-sequence dependencies and modeling global context. Nevertheless, existing Transformer-based approaches, such as those by Demirkıran et al.[22] and Trizna et al.[42], still face challenges in paying sufficient attention to key discriminative features when confronted with multi-class tasks involving high inter-class similarity and ambiguous behavioral boundaries. To address these limitations, the proposed method introduces a gating mechanism within the Transformer framework to construct enhanced semantic representations. By adaptively modeling the global features of API call sequences, the model improves its discriminative capability for high-similarity behavior classes. Overall, under an acceptable detection delay (3617 ms/k), the proposed ap-

proach achieves superior multi-class accuracy (86.16%) and binary accuracy (97.73%) compared to baseline methods (Table 5), demonstrating that the method attains improved detection performance while maintaining efficiency.

Table 5: Performance Comparison with current work on the AliYun dataset

Method	Type	Validation Results		
		Multi ACC (%)	Binary ACC (%)	MTTD (ms/k)
Li et al. [13] (2022)	CNN+RNN-based	83.84	95.39	480
Demirkıran et al. [22] (2022)	Transformer-based	76.82	93.59	3340
Trizna et al. [42] (2024)	Transformer-based	76.24	90.50	3611
Proposed Model	**Transformer-based**	**86.16**	**97.73**	**3617**

To further analyze the model's performance on high-similarity and rare classes, the confusion matrices of all methods under the same fold of cross-validation were compared, as shown in Fig. 10. It should be noted that for rare classes such as *Worm*, which contain only a limited number of samples, all methods face notable difficulties in accurate recognition on the AliYun dataset. This behavior primarily arises from the severe class imbalance and the high overlap of behavioral features across different malware families. Despite these challenges, the proposed method produces a relatively more concentrated overall prediction distribution.



(a) (b) (c) (d)

Figure 10: Comparison of confusion matrices of different methods on the AliYun dataset under one fold of cross-validation. (a) Li et al. [13] CNN+RNN-based. (b) Demirkıran et al. [22] Transformer-based. (c) Trizna et al. [42] Transformer-based. (d) Proposed method.

Furthermore, to verify the robustness of the model under different conditions, perturbation experiments were conducted during testing by introducing 10% high-frequency benign API insertion noise and 3-gram API segment noise. The results are shown in Table 6. It can be observed that after the introduction of noise, the multi-class accuracy and weighted F1-score only slightly decrease, indicating that the model does not rely on simple

memorization of API call patterns but can robustly leverage global semantic features for discrimination. These results further validate the generalization capability and robustness of the proposed method.

Table 6: Robustness Evaluation under Noisy Conditions on the AliYun Dataset

Method / Perturbation	**Multi-class Accuracy (%)**	**Weighted F1-score (%)**
Insert Noise (10%)	85.44	85.24
3-gram Noise (10%)	85.66	85.63
Proposed model	86.16	85.71

5. Limitations and Discussion

Compared to the aforementioned works, this paper achieves effective malware detection solely relying on API call sequences, which enhances the generality of the detection method and addresses the practical application challenge of malware identification and detection due to the high complexity of API call parameters. In the future, this method could be extended to Android malware detection tasks. However, there are still some limitations in this work, which provide ideas for future research directions:

- The proposed method relies on dynamic analysis, requiring the execution of malware samples within a virtual environment or sandbox for a certain period in order to capture their behavioral characteristics. Compared with traditional static detection methods, dynamic analysis exhibits slower detection speed. Nevertheless, the primary motivation for adopting dynamic analysis is to enhance the capability to identify complex malicious behaviors, particularly in the detection of zero-day attacks, obfuscated samples, and malware variants, addressing the challenges posed by advanced persistent threats (APTs) to system and device security.

- The model utilizes API call sequences of up to 512 tokens for semantic modeling, which necessitates the accumulation of sufficient behavioral data before making a prediction. Consequently, certain types of malware, such as ransomware, may introduce a slight detection delay. Given these characteristics, the proposed approach is more suitable for dynamic analysis in cloud-based or local sandbox environments and can serve as a high-precision behavioral analysis module following static or rule-based detection. This enables the method to work synergistically with existing approaches to construct a multi-layered defense system.

- Although this method has been validated on the publicly available Datacon 2019 dataset [32], the timeliness and representativeness of this dataset are limited, and it may not be sufficient to address the latest malware variants. Therefore, we plan to collaborate with security companies to acquire more comprehensive and representative real-world malware samples, thereby enhancing the model's adaptability and generalization ability.

- Although this method demonstrates strong robustness in malware detection tasks, its performance declines when faced with imbalanced data distributions. Future research will focus on addressing the long-tail distribution issue of data and further analyzing the feature differences between different malware families to achieve more accurate family classification. Additionally, we plan to introduce active learning strategies to address the concept drift caused by malware variants.

6. Conclusion

To further enhance the efficiency of malicious software detection, this paper proposes a Windows dynamic malware detection scheme based on enhanced semantic API sequence features. The scheme leveraged the RoBERTa model to fully extract potential correlations within API sequences, thereby enhancing the mining capability of contextual semantic information. Additionally, the gating mechanism was designed to adjust the feature weights between the global and semantic modalities, thereby augmenting the semantic features of API sequences and improving sensitivity to detecting various types of malicious software samples. Furthermore, the use of a multi-head attention mechanism suppresses noise within the enhanced semantic API sequences, improving the model's understanding of semantic and structural information within the sequence features. The proposed method has been validated on multiple publicly available malware datasets, demonstrating its effectiveness and robustness, and it can be extended to multi-classification tasks, showing good applicability and generalization capability.

Traditional malware detection methods often struggle to completely prevent the potential threats posed by malware to endpoint systems. Consequently, before software deployment, it is common practice to execute and analyze the software within a sandbox or virtual environment, in conjunction with implementing multi-layered protection using firewalls. As such, our proposed dynamic detection scheme enables real-time monitoring and analysis

of software behavior in an isolated environment, ensuring timely and effective interception and handling before malicious programs can affect endpoint systems.

Acknowledgement

This work is supported by the National Natural Science Foundation of China Grant No. 62102190.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

References

[1] Av-test, [Online], `https://portal.av-atlas.org/malware` (2023).

[2] G. D. Tizio, M. Armellini, F. Massacci, Software updates strategies: A quantitative evaluation against advanced persistent threats, IEEE Transactions on Software Engineering 49 (3) (2023) 1359–1373.

[3] J. Singh, J. Singh, A survey on machine learning-based malware detection in executable files, Journal of Systems Architecture 112 (2021) 101861.

[4] C. Molloy, J. Banks, S. H. H. Ding, F. Alaca, P. Charland, A. Walenstein, Mecha: A neural-symbolic open-set homogeneous decision fusion approach for zero-day malware similarity detection, IEEE Transactions on Software Engineering 51 (2) (2025) 621–637.

[5] S. Sibi Chakkaravarthy, D. Sangeetha, V. Vaidehi, A survey on malware analysis and mitigation techniques, Computer Science Review 32 (2019) 1–23.

[6] X. Ling, L. Wu, J. Zhang, Z. Qu, W. Deng, X. Chen, Y. Qian, C. Wu, S. Ji, T. Luo, J. Wu, Y. Wu, Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art, Computers & Security 128 (2023) 103134.

[7] R. Muthalagu, J. Malik, P. M. Pawar, Detection and prevention of evasion attacks on machine learning models, Expert Systems with Applications 266 (2025) 126044.

[8] A. A. E. Elhadi, M. A. Maarof, B. I. Barry, H. Hamza, Enhancing the detection of metamorphic malware using call graphs, Computers & Security 46 (2014) 62–78.

[9] R. K. Koppanati, M. Santra, S. K. Peddoju, D 2 4d: Dynamic deep 4-dimensional analysis for malware detection, IEEE Transactions on Information Forensics and Security (2025).

[10] X. Chen, Z. Hao, L. Li, L. Cui, Y. Zhu, Z. Ding, Y. Liu, Cruparamer: Learning on parameter-augmented api sequences for malware detection, IEEE Transactions on Information Forensics and Security 17 (2022) 788–803.

[11] Z. Zhang, P. Qi, W. Wang, Dynamic malware analysis with feature engineering and feature learning, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 34, 2020, pp. 1210–1217.

[12] C. Li, Q. Lv, N. Li, Y. Wang, D. Sun, Y. Qiao, A novel deep framework for dynamic malware detection based on api sequence intrinsic features, Computers & Security 116 (2022) 102686.

[13] C. Li, Z. Cheng, H. Zhu, L. Wang, Q. Lv, Y. Wang, N. Li, D. Sun, Dmalnet: Dynamic malware analysis based on api feature engineering and graph learning, Computers & Security 122 (2022) 102872.

[14] L. Cui, J. Cui, Y. Ji, Z. Hao, L. Li, Z. Ding, Api2vec: Learning representations of api sequences for malware detection, in: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Association for Computing Machinery, New York, NY, USA, 2023, p. 261–273.

[15] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding (2019). arXiv:1810.04805.

[16] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pre-training approach (2019). arXiv:1907.11692.

[17] G. Paolini, B. Athiwaratkun, J. Krone, M. Jie, A. Achille, R. Anubhai, C. N. dos Santos, B. Xiang, S. Soatto, et al., Structured prediction as translation between augmented natural languages, in: ICLR 2021-9th International Conference on Learning Representations, International Conference on Learning Representations, ICLR, 2021, pp. 1–26.

[18] H. Yan, J. Dai, T. Ji, X. Qiu, Z. Zhang, A unified generative framework for aspect-based sentiment analysis, in: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 2021, pp. 2416–2429.

[19] H. Bao, L. Dong, S. Piao, F. Wei, BEit: BERT pre-training of image transformers, in: International Conference on Learning Representations, 2022.

[20] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, D. Roth, Recent advances in natural language processing via large pre-trained language models: A survey, ACM Comput. Surv. 56 (2) (sep 2023).

[21] Z. Xu, X. Fang, G. Yang, Malbert: A novel pre-training method for malware detection, Computers & Security 111 (2021) 102458.

[22] F. Demirkıran, A. Çayır, U. Ünal, H. Dağ, An ensemble of pre-trained transformer models for imbalanced multiclass malware classification, Computers & Security 121 (2022) 102846.

[23] R. Tian, R. Islam, L. Batten, S. Versteeg, Differentiating malware from cleanware using behavioural analysis, in: 2010 5th International Conference on Malicious and Unwanted Software, 2010, pp. 23–30.

[24] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, A. Hamze, Malware detection based on mining api calls, in: Proceedings of the 2010 ACM symposium on applied computing, 2010, pp. 1020–1025.

[25] E. Amer, I. Zelinka, A dynamic windows malware detection and prediction method based on contextual understanding of api call sequence, Computers & Security 92 (2020) 101760.

[26] J. Singh, J. Singh, Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms, Information and Software Technology 121 (2020) 106273.

[27] D. Rabadi, S. G. Teo, Advanced windows methods on malware detection and classification, in: Proceedings of the 36th Annual Computer Security Applications Conference, 2020, pp. 54–68.

[28] B. Ndibanje, K. H. Kim, Y. J. Kang, H. H. Kim, T. Y. Kim, H. J. Lee, Cross-method-based analysis and classification of malicious behavior by api calls extraction, Applied Sciences 9 (2) (2019).

[29] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, M. Yang, Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware, in: Proceedings of the 2020 ACM SIGSAC conference on computer and communications security, 2020, pp. 757–770.

[30] L. Xiaofeng, J. Fangshuo, Z. Xiao, Y. Shengwei, S. Jing, P. Lio, Assca: Api sequence and statistics features combined architecture for malware detection, Computer Networks 157 (2019) 99–111.

[31] X. Li, Z. Li, H. Xie, Q. Li, Merging statistical feature via adaptive gate for improved text classification, Proceedings of the AAAI Conference on Artificial Intelligence 35 (15) (2021) 13288–13296.

[32] Malicious-code-dataset, [Online], `https://github.com/kericwy1337` (2019).

[33] Y. Ki, E. Kim, H. K. Kim, A novel approach to detect malware based on api call sequence analysis, International Journal of Distributed Sensor Networks 11 (6) (2015) 659101.

[34] F. O. Catak, A. F. Yazı, A benchmark api call dataset for windows pe malware classification (2021). arXiv:1905.01999.

[35] A. Cloud, Alibaba cloud malware detection based on behaviors, https://tianchi.aliyun.com/getStart/information.htm?raceId=231694, online; accessed 11-November-2018 (2018).

[36] H. Yang, S. Li, X. Wu, H. Lu, W. Han, A novel solutions for malicious code detection and family clustering based on machine learning, IEEE Access 7 (2019) 148853–148860.

[37] X. Chen, Y. Tong, C. Du, Y. Liu, Z. Ding, Q. Ran, Y. Zhang, L. Cui, Z. Hao, Malpro: Learning on process-aware behaviors for malware detection, in: 2022 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2022, pp. 01–07.

[38] Z. Ding, H. Xu, Y. Guo, L. Yan, L. Cui, Z. Hao, Mal-bert-gcn: Malware detection by combining bert and gcn, in: 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2022, pp. 175–183.

[39] C. Du, Y. Tong, X. Chen, Y. Liu, Z. Ding, H. Xu, Q. Ran, Y. Zhang, L. Meng, L. Cui, Z. Hao, Toward detecting malware based on process-aware behaviors, Security and Communication Networks 2023 (1) (2023) 6447655.

[40] T. Chen, H. Zeng, M. Lv, T. Zhu, Ctimd: Cyber threat intelligence enhanced malware detection using api call sequences with parameters, Computers & Security 136 (2024) 103518.

[41] R. Hou, D. Liu, X. Jin, J. Weng, G. Geng, A malware detection method with function parameters encoding and function dependency modeling, PeerJ Computer Science 11 (2025) e2946.

[42] D. Trizna, L. Demetrio, B. Biggio, F. Roli, Nebula: Self-attention for dynamic malware analysis, IEEE Transactions on Information Forensics and Security 19 (2024) 6155–6167.