frontiers | Frontiers in Physics

TYPE Original Research
PUBLISHED 01 November 2024
DOI 10.3389/fphy.2024.1493209

Check for updates

OPEN ACCESS

EDITED BY
Xuzhen Zhu,
Beijing University of Posts and Telecommunications (BUPT), China

REVIEWED BY
Zhidan Zhao,
Shantou University, China
Lei Gao,
Shandong Agricultural University, China
Chao Fan,
Chengdu University of Technology, China

*CORRESPONDENCE
Yali Wang,
ylwang@szcu.edu.cn

RECEIVED 08 September 2024
ACCEPTED 14 October 2024
PUBLISHED 01 November 2024

CITATION
Xue H, Wang Y and Tang Q (2024) Dynamic analysis of malicious behavior propagation based on feature selection in software network. *Front. Phys.* 12:1493209.
doi: 10.3389/fphy.2024.1493209

Dynamic analysis of malicious behavior propagation based on feature selection in software network

Huajian Xue[1,2], Yali Wang[3]* and Qiguang Tang[4]

[1]College of Mathematics and Computer Science, Tongling University, Tongling, China, [2]Anhui Engineering Research Center Of Intelligent Manufacturing Of Copper-based Materials, Tongling University, Tongling, China, [3]College of Computing Science and Artificial Intelligence, Suzhou City University, Suzhou, China, [4]Zhongyuan Oilfield Oil and Gas Engineering Service Center, Zhongyuan Oilfield Company of SINOPEC, Puyang, China

In the era of big data, the propagation of malicious software poses a significant threat to corporate data security. To safeguard data assets from the encroachment of malware, it is essential to conduct a dynamic analysis of various information propagation behaviors within software. This paper introduces a dynamic analysis detection method for malicious behavior based on feature extraction (MBDFE), designed to effectively identify and thwart the spread of malicious software. The method is divided into three stages: First, variable-length N-gram algorithms are utilized to extract subsequences of varying lengths from the sample API call sequences as continuous dynamic features. Second, feature selection techniques based on information gain are employed to identify suitable classification features. Lastly, recurrent neural networks (RNN) are applied for the classification training and prediction of diverse software behaviors. Experimental results and analysis demonstrate that this approach can accurately detect and promptly interrupt the information dissemination of malicious software when such behavior occurs, thereby enhancing the precision and timeliness of malware detection.

KEYWORDS

recurrent neural networks, information propagation, feature selection, dynamic analysis, software network

1 Introduction

In the information age, business data has become the lifeblood of enterprises, and one of the major risks in business operations is the destruction of commercial data by malicious software. Today, with the high integration of the Internet of Things, big data, and mobile Internet, malicious attacks pose an unprecedented threat to corporate data information. For instance, the "Panda Burning Incense" virus in 2006 infected millions of personal computer users and enterprise local area networks. The Aurora attack in 2010 led to the theft of information data from more than 20 companies worldwide. The ransomware virus attack in 2017 prevented the important servers of hundreds of companies from starting. These malicious software behaviors have stolen or destroyed corporate data assets, causing immeasurable losses to businesses. Whether enterprise information systems can be used normally and safely is an important issue that cannot be ignored. To combat the threat posed by the explosive growth of malicious software to corporate data information systems, researchers have studied the detection of malicious behavior from different perspectives.

Social science researchers mainly conduct qualitative analysis in the detection of malicious behavior, analyzing various risk factors that enterprises face in the context of big data from a macro perspective [1–5]. Natural science researchers mainly study the behavior and characteristics of malicious software through methods such as N-gram, graph theory, and Bayesian classification [6–11], committing to finding a strategy that can quickly detect malicious software, thereby strengthening the risk prevention of corporate data assets. Although these schemes have improved the detection rate of malicious behavior, there are still shortcomings. Methods based on fixed-length N-grams struggle to fully describe the behavior of malicious software. On one hand, different behaviors of malicious software correspond to different sequences of API calls. Moreover, the number of API calls varies as the malicious software performs different operations. On the other hand, malicious software can evade traditional fixed-length API N-gram malware detection methods by inserting independent API calls during execution. Malware detection systems based on graph theory can detect variants of malicious software and have a high detection accuracy. However, these feature extraction methods have limitations. Typically, there are hundreds or thousands of vertices or edges in a program's behavioral call graph. Therefore, constructing a behavioral call graph is relatively difficult.

Based on an in-depth analysis of existing research results, in this paper, we proposes a malicious behavior detection method based on feature extraction(MBDFE), aimed at identifying whether the behavior of software programs contains malicious elements. The work of this paper mainly includes the following two aspects:

- This paper innovatively proposes a software feature selection technique that uses the N-gram algorithm to capture operation codes during the software execution process and employs an information gain calculation method to select the most representative software features. When new software behavior patterns are detected in the enterprise software information system, this feature selection technique can accurately extract the code that reflects its behavioral characteristics.
- This paper transforms the dynamic analysis of malicious behavior into a classification problem, inputs the extracted software feature codes into a recurrent neural network model for processing, and judges whether the software is malicious based on the classification results of the model's software behavior. This method can efficiently identify and warn of potential malicious behavior, providing timely security protection measures for enterprises.

The subsequent chapters of this paper are arranged as follows: Chapter 2 reviews the previous research work in the relevant field; Chapter 3 elaborates on the framework and specific implementation details of the proposed plan; Chapter 4 validates the performance of the proposed plan through extensive experiments; finally, Chapter 5 summarizes the proposed plan and provides a perspective on future research directions.

2 Related works

In this section, we review previous research achievements in malicious behavior detection. The identification of malicious software behavior is an interdisciplinary research direction, where both social science and natural science researchers have conducted in-depth studies on this topic. Social science researchers primarily employ qualitative analysis to examine the various risk factors of corporate risk in the context of big data [1–5], and subsequently propose strategies and countermeasures to address these risks.

Meng Fanfei in literature [1] reviews the development history of the COBIT framework, the concepts and theories related to IT governance and risk management, and applies the content of the COBIT framework to IT governance and enterprise risk management. The paper analyzes the advantages and feasibility of using the COBIT framework from multiple perspectives and proposes some techniques and methods in the application process to facilitate better integration of the COBIT framework into IT governance and risk management by enterprises. Peng Chaoran et al. [2] point out that the construction of domestic enterprise information platforms is lagging, and there are significant security risks in placing the data assets of large enterprises on platforms of foreign giants. They propose from a strategic height the construction of independent enterprise information resource platforms, accounting information standard firewalls, and enterprise information security regulations.

Liu Shangxi et al. in literature [3] use neural network technology to identify corporate tax risks. The paper uses the financial data of 578 enterprises as training samples, derives the characteristics of enterprise risks, and validates them with a sample of 386 enterprises, achieving a final accuracy of 99.8%. Yang Ling [4] constructs a corporate operation risk monitoring classification and grading index system based on a "big data platform". The system obtains real-time monitoring indicator data through a data asset collaborative application platform, realizes real-time risk early warning, emergency linkage, and closed-loop risk management according to preset thresholds, and regularly forms a business risk health index analysis report based on the statistical scores and weights of various indicators, serving as an important reference for leadership decision-making. Zhang Lizhe believes that corporate data asset management faces unprecedented risks in the era of big data. Establishing a comprehensive and reliable financial risk management system and strengthening the prevention of corporate financial risks should become a key research issue for enterprise development. In literature [5], the author analyzes the problems in the development of corporate financial risk management systems, discusses the significance of establishing a sound prevention system, and proposes suggestions for the construction of a financial risk management system model.

Natural science researchers primarily utilize statistical models or machine learning methods [6–11] to study the behaviors and characteristics of malicious software. They are committed to finding a strategy that can rapidly detect malicious software, thereby enhancing the risk prevention of corporate data assets. Wang Rui et al. in literature [6] combine dynamic taint propagation analysis and semantic analysis at the behavioral level to extract key system calls of malicious software, dependencies between calls, and related instruction information, constructing a semantic-based malicious software behavior detection system to detect variants of malicious software.

Sathyanarayan et al. [7] use the static analysis method N-gram to extract the frequency of key API calls from programs, and by leveraging the correlation between malicious software semantics and API calls, construct behavioral signatures for entire families of malicious software through statistical comparison. Fang et al. [8] use dynamic analysis methods to extract API calls, return values, module names, and their frequencies as behavioral features from programs, and establish an integrated machine learning algorithm-based malicious software detection model to detect variants of malicious software. Park et al. [9] construct a Kernel Object Behavioral Graph (KOBG) for each piece of malicious software, and then detect new malicious software by clustering to mine the family's minimum weight common supergraph (Weighted Minimum Common Supergraph, WMinCS).

Ding et al. [10] use dynamic taint technology to construct a system call dependency graph based on the parameter dependency relationships between system calls, and then extract a common behavioral subgraph as a signature for each family of malicious software based on the maximum weight subgraph (maximum weight subgraph, MWS) algorithm to detect variants of malicious software.

Zhang et al. in [11] propose a deep detection method for malware based on behavior chains (MALDC). This method monitors behavior points based on API calls and then constructs behavior chains using the calling sequences of those behavior points at runtime. Finally, a deep detection method based on Long Short-Term Memory (LSTM) networks is used to detect malicious behavior from the behavior chains.

Li et al. in [12]propose a feature fusion, machine learning-based method to detect malicious mining code. Extracts multi-dim features via static and statistical analysis. Uses n-gram, TF-IDF for text feature vectors, selects best via classifier, and fuses with stats for model training.

Amer et al. in [13] attempted to create universal behavior models for malicious and benign processes, leveraging statistical, contextual, and graph mining features to capture API function relationships in call sequences. Generated models show behavior contrast, leading to relational perspective models that characterize process behaviors. Zhan et al. [14] propose an anomaly detection method for adversarial robustness, analyzing behavior units to tackle issues. Behavior units, extracted from related actions executing intentions, hold key semantic info for local behaviors, boosting analysis robustness. Using a multi-level DL model, it learns semantics and context of behavior units to counter local and broad-scale perturbation attacks. Wong et al. [15] use deep learning to pinpoint API calls linked to malware techniques in execution traces. APILI sets up multi-attention between API, resources, and techniques, using a neural net to incorporate MITRE ATT&CK, tactics, and procedures. It uses fine-tuned BERT for embedding and SVD for tech representation, with design tweaks like layering and noise to boost location accuracy. Chen et al. [16] propose a method for Windows malware detection uses deep learning on APIs with added parameters. It rates parameter sensitivity to malware via rules and clustering, then tags APIs by sensitivity. APIs are encoded by merging native and sensitivity embeddings to show security relations. These embeddings are used to train a deep neural network binary classifier for malware. Pektaş et al. [17] employ the API call graph to depict the full spectrum of execution routes accessible to malware while it operates. This graph's embedding is converted into a compact numerical vector feature set for integration into a deep neural network. Following this, the detection of similarities within each binary function is efficiently trained and evaluated. Streamlining security analyst tasks, automating Android malware detection and family classification is crucial. Prior research leveraged machine learning to tackle these challenges. Yet, the growing app count poses a need for a scalable, accurate solution in cybersecurity. Here, Sun et al. in [18] introduce a method enhancing malware and family detection, also cutting analysis time.

Tharani et al. [19] introduces a range of feature categories and a streamlined feature extraction technique for Bitcoin and Ethereum transaction data, considering their interconnections. As per our awareness, no prior research has utilized feature engineering for malicious activity detection. These features' relevance was confirmed with eight classifiers: RF, XG, Silas, and neural networks.

Zou et al. [20] aim to merge the precision of graph-based detection with the scalability of social network analysis for Android malware. We analyze app function call graphs as social networks to find central nodes, then measure their intimacy with sensitive APIs. Our IntDroid tool was tested on a dataset with 3,988 benign and 4,265 malicious samples.

3 The proposed method

3.1 The Overview of MBDFE

The solution proposed in this paper is divided into three steps. The first step is to use variable-length N-gram to extract software behavior feature codes; the second step is to reduce the dimensionality of the feature codes through a feature selection method - information gain; the third step is to train the weights of the recurrent neural network with the features. The framework of the solution is shown in Figure 1:

As shown in Figure 1, the framework of the MBDFE method proposed in this study consists of three core components: feature extraction, feature selection, and software behavior recognition. During the feature extraction phase, the n-gram algorithm is used to extract the software's operation codes as feature codes. In the feature selection phase, the information gain algorithm is employed to select feature codes with higher information content as classification features. In the behavior recognition phase, after the classification features are processed through a recurrent neural network, the resulting software classification probability distribution is obtained, with the category having the highest probability being identified as the actual category of the software. In the proposed MBDFE algorithm, multiple variables are involved, the specific meanings of which are detailed in Table 1.

3.2 The extraction of variable-length N-gram features

The N-Gram model is a statistical probability language model based on the idea of dividing the content of a text into byte-sized

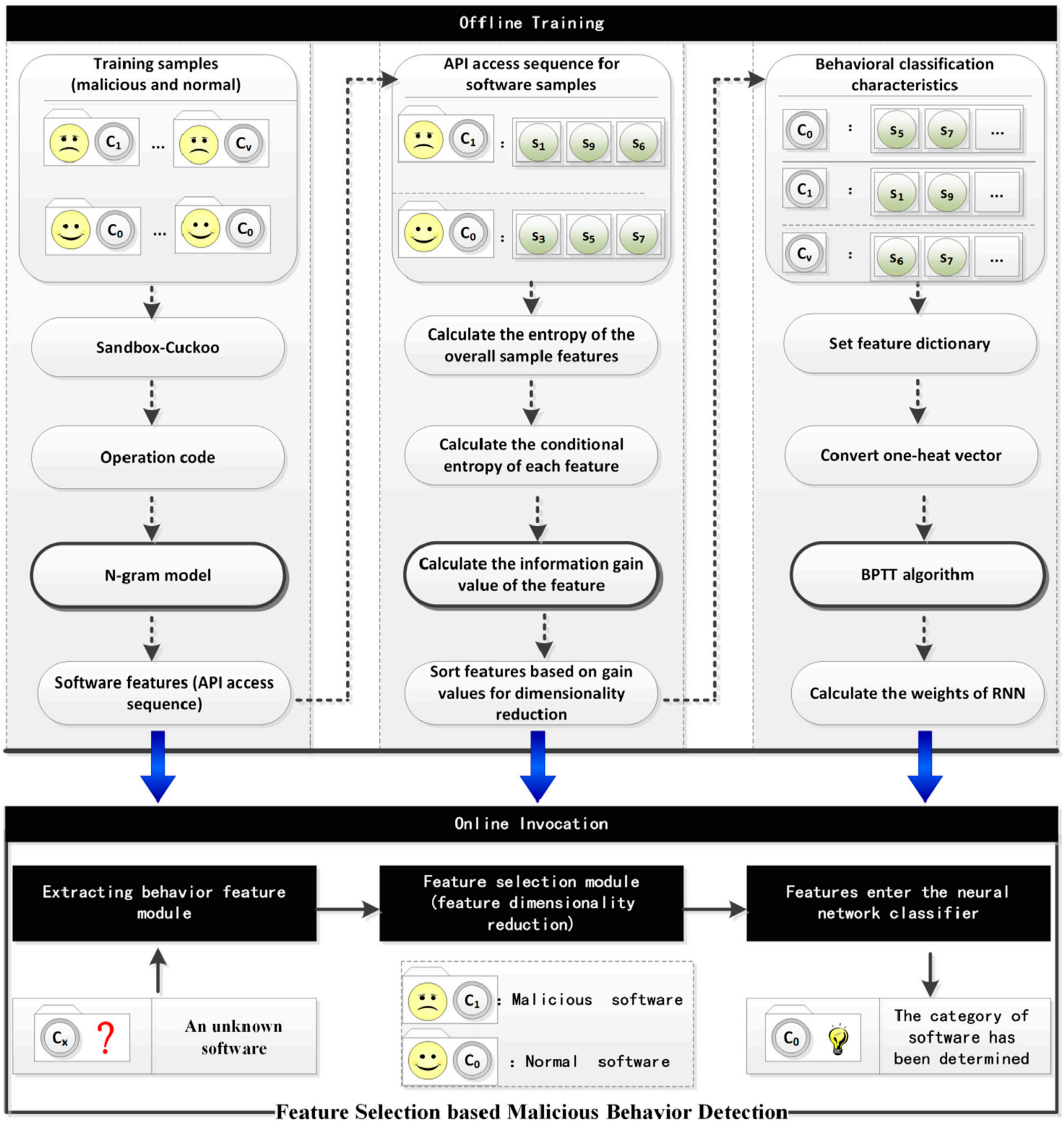
Offline Training
Training samples (malicious and normal)
API access sequence for software samples
Behavioral classification characteristics
Sandbox-Cuckoo
Operation code
N-gram model
Software features (API access sequence)
Calculate the entropy of the overall sample features
Calculate the conditional entropy of each feature
Calculate the information gain value of the feature
Sort features based on gain values for dimensionality reduction
Set feature dictionary
Convert one-heat vector
BPTT algorithm
Calculate the weights of RNN
Online Invocation
Extracting behavior feature module
Feature selection module (feature dimensionality reduction)
Features enter the neural network classifier
An unknown software
Malicious software
Normal software
The category of software has been determined
Feature Selection based Malicious Behavior Detection

FIGURE 1
The overview of MBDFE scheme.

sliding windows of length N, forming a sequence of byte segments of length N. In the field of malicious behavior detection, the N-Gram algorithm extracts the operation codes from the disassembled files of software behavior and converts them into a set of bytes. Then, through a sliding window, a series of n-byte sequences are obtained. These sequences are the feature codes of the software behavior and are a feature extraction method based on the dynamic analysis of malicious software. The variable-length dynamic behavior feature extraction model breaks down the behavioral call sequence of malicious software into different N-grams, performs feature selection on each gram, and then combines them into hyper-grams of varying lengths as the behavioral features of malicious software. This approach aims to detect variants of obfuscated malicious software and improve the accuracy of malicious software detection. The N-Gram primarily utilizes the Markov assumption, and in the field of software malicious behavior detection, the model represents the co-occurrence probability of each byte code and its preceding feature codes. The model is shown in the following formula:

$$p(m_n|m_{n-1}...\mathrm{m}_2\mathrm{m}_1) = \frac{F(\mathrm{m}_1\mathrm{m}_2...m_n)}{F(\mathrm{m}_1\mathrm{m}_2...m_{n-1})} \tag{1}$$

TABLE 1 The variable symbols used in MBDFE scheme.

Variable symbol	Definition of symbols
T	It represents a certain characteristic behavior of the software
t	It represents the value of the characteristic behavior T
H(C)	It represents the entropy of the overall characteristic behavior of the software
H(C|T)	It represents the conditional entropy of feature T
$P(t)$	It represents the proportion of the presence of software behavior T in the entire sequence of software behaviors
$P(C_i \mid t)$	It represents the proportion of software sequences containing software behavior T and belonging to class c in the entire software system containing behavior T behavior sequences
U	It represents the weight between the input layer and the hidden layer in a recurrent neural network
V	It represents the weight between hidden layers
W	It represents the weight between the hidden layer and the output layer
ΔU	It represents the gradient of weight U
ΔV	It represents the gradient of weight V
ΔW	It represents the gradient of weight W
z_t^h	It represents the value of the hidden layer at time t
a_t^h	It represents the activation value of the hidden layer at time t
z_t^o	It represents the value of the output layer at time t
a_t^o	It represents the activation value of the output layer at time t
δ_t^o	It represents the gradient of the output layer at time t
δ_t^h	It represents the gradient of the hidden layer at time t
lr	It represents the learning rate in gradient descent algorithm

in the formula, $p(m_n|m_{n-1}...m_2m_1)$ represents the conditional probability of the operation code m_n given the preceding n−1 items, and $F(m_1m_2...m_n)$ and $F(m_1m_2...m_{n-1})$ represent the frequency of co-occurrence of the operation code sequence. By continuously changing the value of n, the most suitable feature code can be determined based on the derived probability values.

The variable-length N-gram algorithm extracts operation code slices from each software behavior invocation sequence and uses these slices to construct a set of feature codes for software behavior, with the following steps:

- Convert the sample program's invocation behavior into hexadecimal format and match the program's features against a computer virus database.
- Starting from the first position where a match occurs, use a sliding window method to continuously compare backward until no identical features are found. Ensure uniqueness while trying to keep the feature code as short as possible.
- Count the number, or frequency, of features in this byte stream that are included in the virus feature database.
- Set a threshold; when the count of a certain feature exceeds the threshold, add this byte stream to the virus database as a candidate feature code.

3.3 Feature selection by information gain

The feature set obtained through variable-length N-gram segmentation represents a collection of behavior sequences for each software. By training the software's behavior sequences, it is possible to determine the category to which the software sequence belongs (i.e., virus, trojan, ransomware, worm, or normal software access). In this paper, we use recurrent neural networks to classify the behavior sequences of software. However, the importance of features in the software behavior feature vector is not the same in the classification system. To select representative features and improve classification efficiency, it is necessary to quantify the importance of software behavior features. This paper uses Information Gain (IG) to measure the software behavior features. In Information Gain, the criterion is how much information the feature can bring to the classification system; the more information it brings, the more important the feature is.

In the classification system, when a software behavior feature T can be composed of multiple classes (for example, registry access behavior, which could be either normal or trojan behavior), the calculation of conditional entropy needs to consider all its possible values. In the software system, the specific value of the software feature behavior T is set as t. Generally, the values of t are t (indicating that t occurs) and not t (indicating that t does not occur). At this point, the conditional entropy of the behavior feature T is as follows:

$$H(C\,|\,T) = P(t)H(C\,|\,t) + P(\bar{t})H(C\,|\,\bar{t}) \tag{2}$$

in the formula, $P(t)$ represents the proportion of the presence of software behavior T in the entire sequence of software behaviors, and indicates the proportion of software sequences that contain behavior T and belong to class C_i among all software behavior sequences in the software system that include behavior T. Similarly, $P(\bar{t})$ represents the proportion of sequences without behavior T in the entire sequence of software behaviors, and $P(C_i\,|\,\bar{t})$ denotes the proportion of software behavior sequences that do not contain behavior T and belong to class C_i among all software behavior sequences in the software system that do not include behavior T. The entropy of $H(C\,|\,t)$ and $H(C\,|\,t)$ shown in the following formula:

$$\begin{aligned} H(C\,|\,t) &= -\sum_{i=1}^{n} P(C_i\,|\,t)\log_2 P(C_i\,|\,t) \\ H(C\,|\,\bar{t}) &= -\sum_{i=1}^{n} P(C_i\,|\,\bar{t})\log_2 P(C_i\,|\,\bar{t}) \end{aligned} \tag{3}$$

The information gain of software behavior feature T is the difference between the entropy of the entire software behavior and the conditional entropy of software feature T. The formula is as follows:

$$\begin{aligned} IG(T) &= H(C) - H(C\,|\,T) \\ &= -\sum_{i=1}^{n} P(c_i).\log_2 P(c_i) \\ &\quad + \sum_{i=1}^{n} P(C_i\,|\,t)\log_2 P(C_i\,|\,t) + \sum_{i=1}^{n} P(C_i\,|\,\bar{t})\log_2 P(C_i\,|\,\bar{t}) \end{aligned} \tag{4}$$

3.4 Utilizing recurrent neural networks for malicious behavior detection

Before training a neural network, it is necessary to first randomly generate the three weights of the neural network: the weight U, the weight W and the weight V. When the behavioral feature vector is input into the network, the forward propagation is as shown in the following formula:

$$\begin{aligned} z_t^h &= x_t \cdot U + a_{t-1}^h \cdot W \\ a_t^h &= \tanh\left(x_t \cdot U + a_{t-1}^h \cdot W\right) \end{aligned} \tag{5}$$

where z_t^h refers to the value of the hidden layer at time t, a_t^h represents the activation value of the hidden layer at time t, x_t represents the one-hot vector of the API access list at time t, and $\tanh()$ is the activation function of the hidden layer. z_t^o represents the value of the output layer at time t, a_t^o represents the activation value of the output layer at time t, and $\mathrm{softmax}()$ is the activation function of the output layer. The forward propagation formula for the output layer is as follows:

$$\begin{aligned} z_t^o &= a_t^h \cdot V \\ a_t^o &= \mathrm{softmax}\left(a_t^h \cdot V\right) \end{aligned} \tag{6}$$

In this study, the input of RNN network [21] is the sequence of API accesses for each software, with the API access sequence length matching the training time series. After each time step's forward propagation calculates the output layer's activation, these values are collected into a list. Once the forward propagation for the entire sequence is complete, the cross-entropy is used to compute the error between the output layer's activations and the true output labels at each time step. This error is then backpropagated through time to each layer to compute the gradients, which are essential for updating the weights. The cross-entropy loss function is expressed as:

$$\mathrm{Loss} = -\sum_{i=1}^{L_1} y_t(i) * \ln a_t^o(i) \tag{7}$$

where L_1 is the length of the one-hot vector, y_t represents the category to which the input feature sequence belongs at time t, and the error of the output layer at each moment can be obtained through Formula 7. The output layer error can be used to calculate the gradient about the output layer through the chain rule. Define δ_t^o as the gradient of the output layer at time t, δ_t^h as the gradient of the hidden layer at time t, $\frac{\partial \mathrm{Loss}_t}{\partial a_t^o}$ as the gradient of the loss function at time t on the activation value of the output layer, and $\frac{\partial a_t^h}{\partial z_t^h}$ as the gradient of the activation value of the hidden layer at time t on the hidden layer. The gradients of U, V, and W are defined as ΔU, ΔV, and ΔW, and are solved as follows:

$$\begin{aligned} \Delta U &= \frac{\partial \mathrm{Loss}_t}{\partial a_t^o} \cdot \frac{\partial a_t^o}{\partial z_t^o} \cdot \frac{\partial z_t^o}{\partial a_t^h} \cdot \frac{\partial a_t^h}{\partial z_t^h} \cdot \frac{\partial z_t^h}{\partial U} + \frac{\partial \mathrm{Loss}_{t+1}}{\partial a_{t+1}^o} \cdot \frac{\partial a_{t+1}^o}{\partial z_{t+1}^o} \cdot \frac{\partial z_{t+1}^o}{\partial a_{t+1}^h} \cdot \frac{\partial a_{t+1}^h}{\partial z_{t+1}^h} \\ &\quad \cdot \frac{\partial z_{t+1}^h}{\partial a_t^h} \cdot \frac{\partial a_t^h}{\partial z_t^h} \cdot \frac{\partial z_t^h}{\partial U} \\ &= \delta_t^h \cdot \frac{\partial z_t^h}{\partial U} = \delta_t^h \cdot \frac{\partial\left(x_t \cdot U + a_{t-1}^h \cdot W\right)}{\partial U} = \delta_t^h \cdot x_t \\ \Delta V &= \frac{\partial \mathrm{Loss}_t}{\partial a_t^o} \cdot \frac{\partial a_t^o}{\partial z_t^o} \cdot \frac{\partial z_t^o}{\partial V} = \delta_t^o \cdot \frac{\partial z_t^o}{\partial V} = \delta_t^o \cdot \frac{\partial\left(a_t^h \cdot V\right)}{\partial V} = \delta_t^o \cdot a_t^h \\ \Delta U &= \frac{\partial \mathrm{Loss}_t}{\partial a_t^o} \cdot \frac{\partial a_t^o}{\partial z_t^o} \cdot \frac{\partial z_t^o}{\partial a_t^h} \cdot \frac{\partial a_t^h}{\partial z_t^h} \cdot \frac{\partial z_t^h}{\partial W} + \frac{\partial \mathrm{Loss}_{t+1}}{\partial a_{t+1}^o} \cdot \frac{\partial a_{t+1}^o}{\partial z_{t+1}^o} \cdot \frac{\partial z_{t+1}^o}{\partial a_{t+1}^h} \cdot \frac{\partial a_{t+1}^h}{\partial z_{t+1}^h} \\ &\quad \cdot \frac{\partial z_{t+1}^h}{\partial a_t^h} \cdot \frac{\partial a_t^h}{\partial z_t^h} \cdot \frac{\partial z_t^h}{\partial W} \\ &= \delta_t^h \cdot \frac{\partial z_t^h}{\partial U} = \delta_t^h \cdot \frac{\partial\left(x_t \cdot U + a_{t-1}^h \cdot W\right)}{\partial W} = \delta_t^h \cdot a_{t-1}^h \end{aligned} \tag{8}$$

The weight update of the network is the initial weight updated by the gradient descent method. Before updating, it is necessary to first calculate the cumulative update value of the weight gradient. The initial values of ΔU, ΔV, and ΔW are 0 matrices, and the matrix dimensions are consistent with the dimensions of U, V, and W. During each time step of the training feature sequence, ΔU, ΔV, and ΔW are accumulated and updated, and the cumulative update equation is as follows:

$$\begin{aligned} \Delta U &= \Delta U + \delta_t^h \cdot x_t \\ \Delta V &= \Delta V + \delta_t^o \cdot a_t^h \\ \Delta W &= \Delta W + \delta_t^h \cdot a_{t-1}^h \end{aligned} \tag{9}$$

When the feature sequence is not yet trained, the three weights are shared throughout the time sequence training process and will not be updated. Once the feature sequence of the entire software is trained, U, V, and W can be updated by the gradient descent method. The weight update equation is as follows:

$$\begin{aligned} U &= U - \mathrm{lr}*\Delta U \\ V &= V - \mathrm{lr}*\Delta V \\ W &= W - \mathrm{lr}*\Delta W \end{aligned} \tag{10}$$

When a software behavior is running, our proposed scheme can quickly determine the probability of this software belonging to various categories based on the sequence vector of software behavior and the weight vector, taking the category with the highest probability as the true category of this software, thereby quickly predicting whether the software has malicious behavior.

4 Experiments

4.1 Dataset

The experimental dataset includes 172 benign executable programs and 457 malicious software samples (across 4 types of malware: Trojans, worms, script viruses, and system viruses). All of these samples are Windows Portable Executable (PE) files, including formats such as EXE, DLL, OCX, SYS, and COM. The malicious samples were randomly selected from the malicious software sample set downloaded from the VX Heaven website, while the benign samples were collected from clean Windows systems and the school FTP website as good executable programs.

In order to obtain the behavioral characteristics of these programs, we selected the open-source dynamic analysis tool Cuckoo Sandbox [22]. The Cuckoo Sandbox mainly analyzes file types such as Windows executable files, DLL files, MS Office files, compressed files, etc. It can automatically analyze the dynamic behaviors of executable programs, including process behavior, network behavior, and file behavior. In our experiment, the architecture of the Cuckoo Sandbox primarily involves running the main Cuckoo program on the host machine (the host system is Ubuntu Server 16.10), with multiple guest machines (the environments required for the execution of malicious and benign programs are Windows series operating systems) connected to the host via a virtual network. Each guest machine has a Cuckoo Agent program that acts as a monitoring agent. For data storage security, we have connected a workstation to the Cuckoo host to back up the generated analysis reports and process data. Additionally, analysis can be conducted remotely via the internet by accessing the host. The structure of the Cuckoo Sandbox is shown in Figure 2:

4.2 Experimental evaluation criteria

The experiments in the paper evaluate the performance of the proposed scheme based on accuracy, recall, and F1-measure. For malicious software behavior prediction, accuracy refers to the proportion of predicted malicious samples that are truly malicious, while recall is the proportion of malicious samples in the dataset that we correctly identify as malicious through our scheme. We define TP as the number of positive samples in the dataset predicted as positive, FN as the number of positive samples in the dataset predicted as negative, FP as the number of negative samples predicted as positive, and TN as the number of negative samples predicted as negative. The evaluation criteria are shown in the following formulas:

$$\begin{aligned} Accuacy &= \frac{TP}{TP + FP} \\ \mathrm{Recall} &= \frac{TP}{TP + FN} \\ F1 - measure &= \frac{2Accuacy^{*}\mathrm{Recall}}{Accuacy + \mathrm{Recall}} \end{aligned} \tag{11}$$

4.3 Experimental results and analysis

In this experiment, the hardware configuration utilized is as follows: the central processing unit (CPU) is an Intel Xeon Gold 6234, equipped with 32 GB of memory, a 2 TB hard disk drive, and an NVIDIA GeForce RTX 3080Ti graphics card. In the experiment, we divided the dataset into two parts, with 80% as the training set and 20% as the test set. Regarding the parameter settings for our scheme, we conducted the following experiments:

For the parameter n in the N-gram scheme, the range of values from 1 to 5 was tested, and the results are shown in the figure below:

From Figure 3, it can be observed that when the value of n is 4, the predictive performance is optimal, with the F1-measure of the scheme reaching 0.8627. However, with each increment of n in the n-gram scheme, the number of behavior features increases exponentially, and the time cost also rises. We can see in the chart that when n increases from 3 to 4, the time cost jumps from 386.24 s to 589.84 s. Considering that when n is taken as 3, the F1-measure of the proposed scheme for identifying malicious behavior reaches 0.8435, which is only about 2% less than the performance when n is 4, we opt for n to be 3 after a comprehensive assessment.

To validate the efficiency of the proposed solution, we first determined the parameters of the model, which consist of four elements: the number of iterations for the model, the learning rate of the model, the dimensionality of the hidden layer, and the number of features. By adjusting the aforementioned parameters, the model's mean loss and accuracy also continuously change, as shown in Figure 4.

From Figure 4A, it can be observed that when the number of iterations reaches 1800, the mean loss of the model is at its lowest, indicating that the model has converged at this point. Figures 4B, C show that when the learning rate is set to 0.04 and the dimensionality of the hidden layer is 220, the model performs the best, with a mean loss value of 0.23416. Regarding the selection of features, we defined the range of feature selection from 40 to 260, increasing by 20 each time, resulting in a series of accuracy values as depicted in Figure 4D. It can be seen from the figure that when the number of feature values is 200, the model's performance is optimal, with an accuracy value of 94.27%. When the number of feature values continues to increase, the model's performance remains essentially unchanged.

In terms of performance comparison, we compared our proposed scheme MBDFE with RNN and Naive Bayes. By varying the density of the dataset, the algorithm's running time and performance also change continuously, as shown in Figure 5:

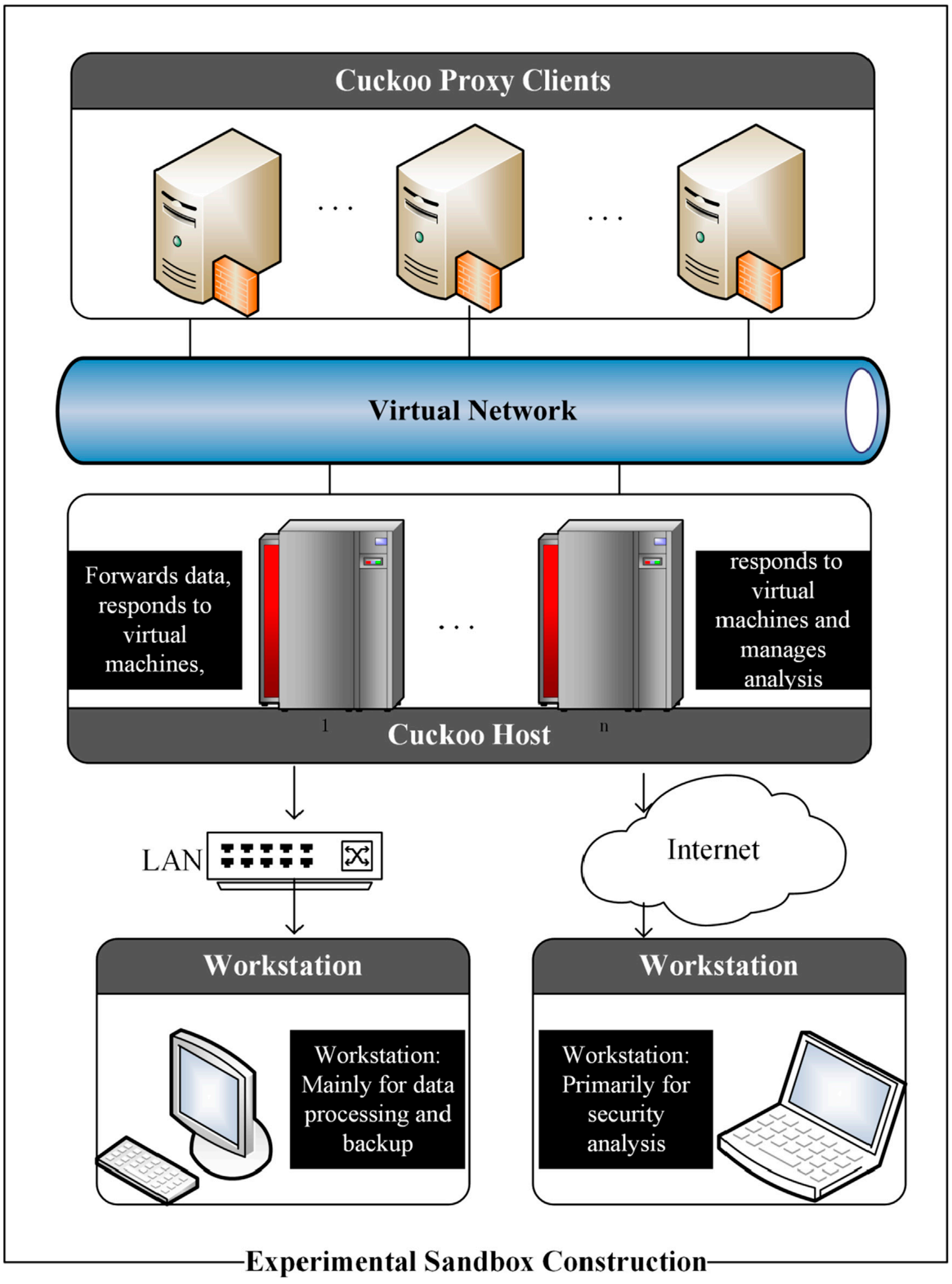
Cuckoo Proxy Clients
. . .
. . .
Virtual Network
Forwards data, responds to virtual machines,
. . .
responds to virtual machines and manages analysis
1
Cuckoo Host
n
LAN
Internet
Workstation
Workstation: Mainly for data processing and backup
Workstation
Workstation: Primarily for security analysis
Experimental Sandbox Construction

FIGURE 2
The structure of cuckoo sandbox.

From Figure 5B, it can be observed that the running times of the models vary. MBDFE and RNN require training the weights of the neural network, while the Bayesian method only needs to calculate the class probabilities for each feature, thus consuming relatively less time. MBDFE, as an RNN model based on feature selection, has an advantage in training time over traditional RNNs. As shown in the figure, when the data density reaches 80%, MBDFE's training time is 1813.15 s, compared to 2876.56 s for RNN, saving 36.97% in time.

Examining Figure 5A reveals that when the data density is less than 50%, MBDFE's performance is inferior to the Bayesian

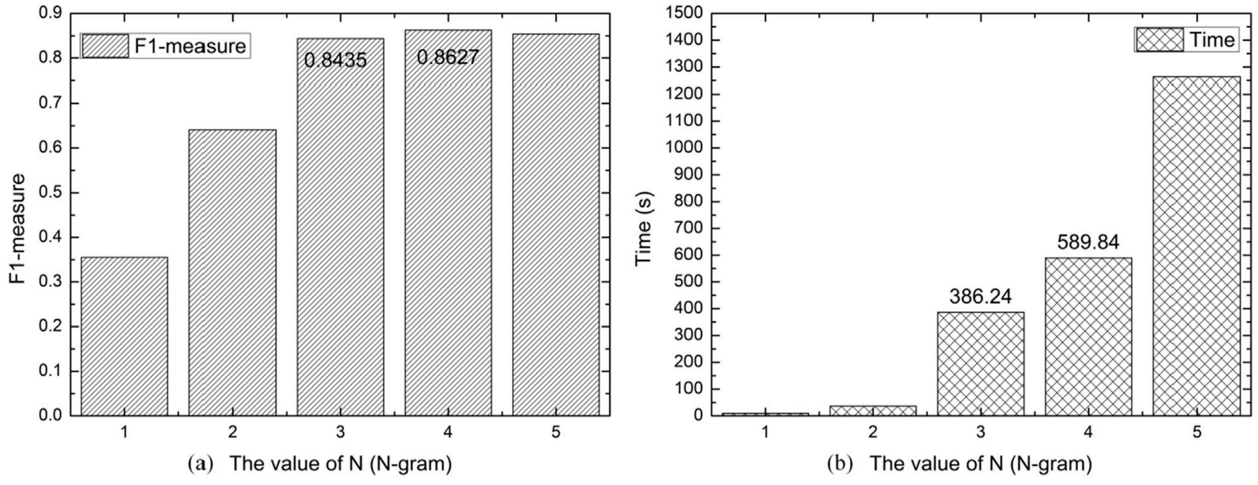

FIGURE 3
The Parameter of N-gram. **(A)** Illustrates the impact of the value of n in n-gram on the performance of the algorithm, and **(B)** shows the effect of the value of n in n-gram on the training time of the algorithm.

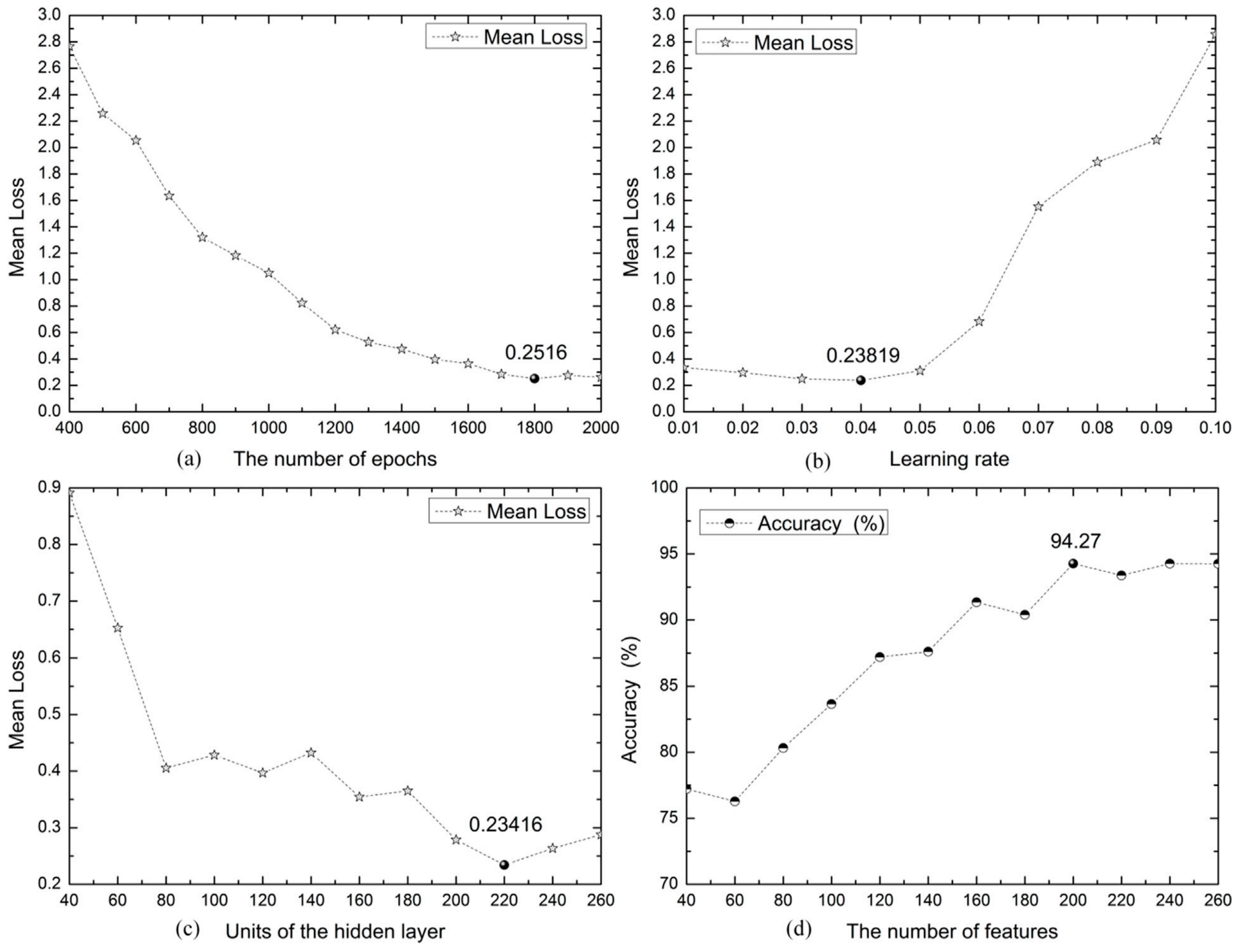

FIGURE 4
The Parameters of MBDFE. **(A)** Demonstrates the impact of the number of parameter iterations on algorithm performance, **(B)** shows the effect of the learning rate on algorithm performance, **(C)** illustrates the influence of the hidden layer dimensions on algorithm performance, and **(D)** presents the impact of the number of feature selections on algorithm performance.

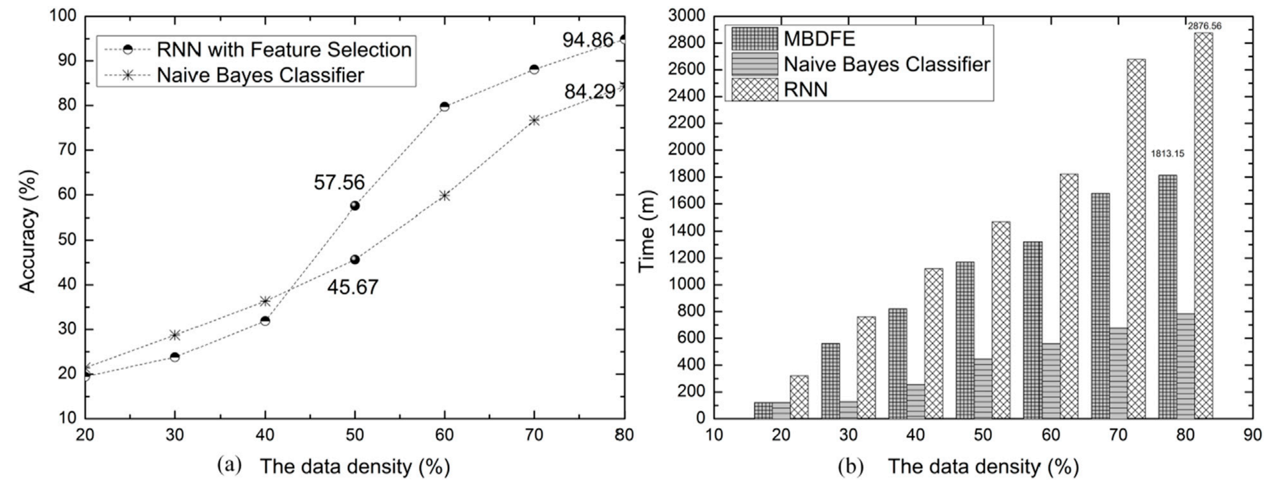

FIGURE 5
The Performance Comparison of Algorithms. **(A)** Presents a comparison of classification performance between the RNN with feature selection proposed in the MBDFE and the Naive Bayes algorithm. **(B)** Illustrates a comparison of algorithmic running times among the MBDFE, the RNN without feature selection, and the Naive Bayes algorithm.

classifier. This is because at lower data densities, the weights of the RNN model within MBDFE are not fully trained, leading to decreased classification performance. However, once the data density exceeds 50%, MBDFE's performance begins to surpass that of the Bayesian classifier. This is due to the Naive Bayes [23] model's assumption of feature independence, which is often not the case in practice, especially with malicious software behaviors that tend to be sequential, limiting the performance of the Bayesian model. In contrast, the RNN in MBDFE can handle sequential data, resulting in better classification performance after full training. Particularly at a data density of 80%, MBDFE achieves an accuracy rate of 94.86%, which is a 13% improvement over the Bayesian model's accuracy rate of 84.29%.

5 Conclusion and future work

This paper explores how to reduce the risk of corporate data assets being compromised by malicious activities and proposes a machine learning technique based on feature selection to identify malicious behaviors within software. The technique primarily uses feature selection algorithms to identify key features of software operation and applies them to a recurrent neural network classifier to determine whether the software's behavior is malicious. Experimental results show that compared to existing algorithms, this approach has improved accuracy in identification.

Although the proposed solution in this paper provides some reference value for malicious behavior detection, no technical solution can predict all malicious behaviors once and for all. Malware attackers will continuously change their attack methods, seeking vulnerabilities in defense systems. Future research should not only enhance the technology for identifying malicious behaviors but also formulate corresponding strategies at the management level based on the development trends of malicious behavior prediction methods. Our future research direction is to combine the engineering methods of natural sciences with the management methods of social sciences to propose an integrated solution for more effective detection and defense against malicious behaviors.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

HX: Methodology, Writing–original draft, Writing–review and editing. YW: Formal Analysis, Writing–review and editing. QT: Software, Writing–review and editing.

Funding

The author(s) declare that financial support was received for the research, authorship, and/or publication of this article. 1. The project of enterprise temporary job of Science and engineering teachers from colleges and universities of Anhui Province (2024jsqygz104). 2. The Project of Artificial intelligence serving the characteristics of ten emerging industries of Anhui province (2023sdxx078). 3. Research on the Teaching Reform of "Software Development and Practice" Course Based on OBE Education Concept under the Open Source Software Ecology of Tongling University (2023xj022).

Acknowledgments

In this study, we utilized the Kimi artificial intelligence technology, version 1.0, 2024, developed by Moonshot AI Co., Ltd., to meticulously check the grammar of our article. The advanced algorithms and user-friendly interface of Kimi greatly facilitated our workflow, ensuring the accuracy and professionalism of the article's grammar. We are particularly grateful for the technical support and resources provided by Moonshot AI Co., Ltd., which enabled our research work to proceed smoothly. For more information about Kimi technology, you can visit the official website: https://kimi.moonshot.cn.

Conflict of interest

Author QT was employed by Zhongyuan Oilfield Company of SINOPEC.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

1. Fanfei M. Research on IT governance and risk management based on COBIT framework. *Shanghai Business* (2021) (1) 3.

2. Peng C. Risk factors and prevention measures of accounting informatization in the big data era. *Fiscal Res* (2014) 000(004):73–6. doi:10.19477/j.cnki.11-1077/f.2014.04.020

3. Liu S, Sun J. Big data thinking: application in tax risk management. *Econ Res Reference* (2016) (9) 19–26. doi:10.16110/j.cnki.issn2095-3151.2016.09.005

4. Yang L. Construction of enterprise operational risk control system based on "big data platform". *Econ Management (Digest Edition)* (2017) 38–9.

5. Zhang L. Construction of enterprise financial risk management system model based on big data. *China Management Informationization* (2017) (17) 2.

6. Rui W, Dengguo F, Yi Y, PuRui S. Semantic-based malicious code behavior feature extraction and detection method. *J Softw* (2012) 2:206–11. doi:10.3724/SP.J.1001.2012.03953

7. Sathyanarayan VS, Kohli P, Bruhadeshwar P. Signature generation and detection of malware families. In: *Proc. of the 13th Australia Conference on International Security and Privacy*, 5107. Berlin: Springer Press (2008). p. 336–349.

8. Ying F, Bo Y, Yong T, Liu L, Zexin L, Yi W, et al. A new malware classification approach based on malware dynamic analysis. In: *Proc. Of australasian conference on information security and privacy ACISP*. Berlin: Springer (2017). p. 173–89.

9. Park Y, Reeves DS, Stamp M. Deriving common malware behavior through graph clustering. *Comput and Security* (2013) 39:419–30. doi:10.1016/j.cose.2013.09.006

10. Ding Y, Xia X, Chen S, Li Y. A malware detection method based on family behavior graph. *Comput and Security* (2018) 73:73–86. doi:10.1016/j.cose.2017.10.007

11. Zhang H, Zhang W, Lv Z, Sangaiah AK, Huang T, Chilamkurti N. MALDC: a depth detection method for malware based on behavior chains. *World Wide Web* (2020) 23(2):991–1010. doi:10.1007/s11280-019-00675-z

12. Li S, Jiang L, Zhang Q, Wang Z, Tian Z, Guizani M. A malicious mining code detection method based on multi-features fusion. *IEEE Trans Netw Sci Eng* (2022) 10(5): 2731–9. doi:10.1109/tnse.2022.3155187

13. Amer E, Zelinka I, El-Sappagh S. A multi-perspective malware detection approach through behavioral fusion of api call sequence. *Comput and Security* (2021) 110:102449. doi:10.1016/j.cose.2021.102449

14. Zhan D, Tan K, Ye L, Yu X, Zhang H, He Z. An adversarial robust behavior sequence anomaly detection approach based on critical behavior unit learning. *IEEE Trans Comput* (2023) 72:3286–99. doi:10.1109/tc.2023.3292001

15. Wong GW, Huang YT, Guo YR, Sun Y, Chen MC. Attention-based API locating for malware techniques. *IEEE Trans Inf Forensics Security* (2023) 19:1199–212. doi:10.1109/tifs.2023.3330337

16. Chen X, Hao Z, Li L, Cui L, Zhu Y, Ding Z, et al. Cruparamer: learning on parameter-augmented api sequences for malware detection. *IEEE Trans Inf Forensics Security* (2022) 17:788–803. doi:10.1109/tifs.2022.3152360

17. Pektaş A, Acarman T. Deep learning for effective Android malware detection using API call graph embeddings. *Soft Comput* (2020) 24:1027–43. doi:10.1007/s00500-019-03940-5

18. Sun B, Takahashi T, Ban T, Inoue D. Detecting android malware and classifying its families in large-scale datasets. *ACM Trans Management Inf Syst (Tmis)* (2021) 13(2): 1–21. doi:10.1145/3464323

19. Tharani JS, Hóu Z, Charles EYA, Rathore P, Palaniswami M, Muthukkumarasamy V. Unified feature engineering for detection of malicious entities in blockchain networks. *IEEE Trans Inf Forensics Security* (2024) 19:8924–38. doi:10.1109/tifs.2024.3412421

20. Zou D, Wu Y, Yang S, Chauhan A, Yang W, Zhong J, et al. IntDroid: android malware detection based on API intimacy analysis. *ACM Trans Softw Eng Methodol (Tosem)* (2021) 30(3):1–32. doi:10.1145/3442588

21. Gao T, Duan L, Feng L, Ni W, Sheng QZ. A novel blockchain-based responsible recommendation system for service process creation and recommendation. *ACM Trans Intell Syst Technology* (2024) 15:1–24. doi:10.1145/3643858

22. Niveditha S, Rr P, Sathya K, Shreyanth S, Subramani N, Deivasigamani B, et al. Predicting malware classification and family using machine learning: a Cuckoo environment approach with automated feature selection. *Proced Computer Sci* (2024) 235:2434–51. doi:10.1016/j.procs.2024.04.230

23. Verma G, Sahu TP. A correlation-based feature weighting filter for multi-label Naive Bayes. *Int J Inf Technology* (2024) 16(1):611–9. doi:10.1007/s41870-023-01555-6