EAI Endorsed Transactions
on AI and Robotics

Research Article EAI.EU

A malware detection method based on LLM to mine semantics of API

Ronghao Hou[1], Xiaoping Tian[2], Guanggang Geng[1]

[1]School for Cyberspace Security, Jinan University, Guangzhou, Guangdong, China
[2]Center of Information and Network Technology, Beijing Normal University, Beijing, China

Abstract

In recent years, the application of the LLM model has played an increasing role in more and more fields, including network security. Some attackers exploit LLM to generate malicious code, craft phishing emails, and analyze software vulnerabilities. It also inspires us to utilize LLM to maintain network security. In previous research on malware detection, feature engineering often relied heavily on expert analysis, making the process both challenging and resource-intensive, especially given the rapid evolution and constant updates of malware. Therefore, we propose a malware detection method for intrinsic semantics. The method first designs an API intrinsic semantic feature encoder, which extracts intrinsic semantic features from API names and Microsoft's official API definitions based on LLM-based prompt engineering and sentence embedding techniques. Then the API co-occurrence feature encoder is designed, which mines the contextual co-occurrence features of API from API call sequences based on the word2vec. The API semantic features and API co-occurrence features are combined to improve the malware detection performance. Also, it uses TCN-GRU to capture dependencies between API calls. Results on several public datasets show that our method achieves better performance than other methods, and in addition, ablation study results demonstrate the important role of intrinsic semantics in malware detection algorithms.

Received on 10 March 2025; accepted on 17 April 2025; published on 07 May 2025

Keywords: malware detection, API sequence, deep learning, feature engineer

doi:10.4108/airo.8880

1. Introduction

Recently, the development of malware has shown a trend of sophistication and diversification. With the popularization of the Internet and the acceleration of the digitalization process, the means of cyber-attacks continue to evolve, and malware has become one of the main weapons of cybercriminals. Current malware trends reflect the increasing use of social engineering techniques, zero-day exploit attacks, and ransomware by attackers for financial gain [1, 2]. In addition, the widespread adoption of the Internet of Things (IoT) has provided new attack surfaces for malware, especially the proliferation of botnets and mining Trojans, exposing enterprise networks to unprecedented threats. Meanwhile, attackers make malware detection and response more difficult through encrypted communication, fuzzing techniques, and multi-layered attack chains.

At the same time, the application of Large Language Modeling (LLM) in the field of cyber-attacks has gradually attracted attention. Generalized language models can increase the efficiency and accuracy in text generation [3]. Attackers use LLM to generate highly realistic phishing emails for social engineering attacks, automate vulnerability mining and exploitation, generate and obfuscate malicious code, write complex attack scripts, and organize intelligence information [4]. These capabilities enable attackers to conduct more precise, stealthy, efficient threats, while significantly lowering the technical threshold and increasing the level of automation of attacks, posing greater defense challenges to network security.

Application Programming Interface (API) gives developers access to the functionality of a software application or service without exposing internal

*Xiaoping Tian. Email: txp@bnu.edu.cn

implementation details [5]. Operating systems provide a wide variety of API for applications to manage hardware. For applications, the API provided by the operating system plays a key role in their development [6]. Malware often needs to directly or indirectly utilize OS-provided API to achieve specific functionality. For example, worms usually utilize file system API to find, copy, and create files, and network API to establish connections, transfer data, and communicate with other systems; Trojans usually also use remote execution API to execute code on other systems, etc.; and ransomware utilizes file system API to obtain files and encryption API to encrypt files.

Currently, many efforts have been made to apply artificial intelligence to analyze API call sequences to enhance the accuracy of malware detection. Compared with complete code analysis, analysis of API call sequences is relatively lightweight, making real-time detection more efficient. In addition, API call sequence analysis focuses on the interaction between interactive programs and operating systems rather than specific code implementations, thus helping to detect unknown and unknown malware variants. Importantly, the API provides dynamic information while the program is running, allowing detection systems to observe and analyze malicious behavior in real-time. Some researchers use machine learning algorithms, such as K-Nearest Neighbor (KNN), Naive Bayes (NB), Decision Tree (DT), Support Vector Machine (SVM), Random Forest (RF), etc., to analyze API call sequences [7–11]. There are also researchers focusing on improving the accuracy of malware detection by adopting deep learning methods for feature extraction [12–18]. However, there are still two factors that limit the effectiveness of API-based malware detection.

On the one hand, some of the current malware detection efforts rely too much on manual analysis by network security experts. Security experts need to use static and dynamic analysis tools to decompile, behaviorally analyze, and sandbox test malware samples in order to get the behavioral characteristics of the malware. We hope to address the difficulties in this with deep learning methods, but deep learning in itself still requires the help of security experts. Thanks to the application of LLM by attackers, we are reminded that we can use LLM to optimize this difficulty.

On the other hand, no attention has been paid to the actual meaning of API. Current research usually adopts the idea of word2vec, which considers API calls as individual words, learns the co-occurrence phenomenon between API to model the potential relationship between words, and finally obtains word vectors. But they seldom consider the rich and deep meanings of API themselves, which are embedded in the names and their meanings of API, which are related to the behaviors of the API. For example, CreateFile is often used to create a new file; ReadFile is often used to read a specified number of bytes from a file handle and store the data in a buffer; OpenProcess can be used to get a handle to a specific process for subsequent operations. Such intrinsic semantics can help us detect malware.

In this paper, we propose new methods to detect malware using API intrinsic semantics. First, we propose a data enhancer based on LLM, which commands LLM to supplement and enhance API semantic data for us through Prompt engineering; then we design the API intrinsic semantic feature encoder. API intrinsic semantic features are extracted from both the API name and API description. For API names, using the data enhancer, we extract API operations and operation objects from the name of the API and model these data from the security perspective as API name features. For API description, which contains the expected behaviors, purposes, and effects of various operations in the API, the API description provided by Microsoft is crawled by a crawler, and the description is supplemented and enriched by using LLM, and the sentence embedding of the API description is extracted by using a deep learning model as the API description feature. The two parts of features are combined as API intrinsic semantic features. At the same time, using the API co-occurrence feature encoder, the word2vec idea is used to view API calls as individual words, learn the co-occurrence phenomenon between API to model the potential relationship of words, and finally get the API co-occurrence features. Finally, the deep neural network is used to mine the dependencies between multi-dimensional features of API calls to detect malware. We have conducted experiments on several publicly available large-scale malware detection datasets, and the experimental results demonstrate that the proposed method achieves better performance than other state-of-the-art methods.

The contributions in this paper are as follows:

- To reduce the consumption of manual analysis, we introduce the LLM-based data enhancer: Using Prompt Engineering, LLM is utilized to supplement and enhance API semantic data to replace the work of experts who deeply analyze the functional roles of different API and extract the features embedded within the API.
- To extract the intrinsic semantics of the API, we proposed the API intrinsic semantic Feature Encoder: Extract API intrinsic semantic features from API names and API descriptions. For the API name, the data enhancer is used to extract the operations and operation objects of the API from the name, and these data are modeled from a security perspective as API name features. For API description, which contains

the expected behaviors, purposes, and effects of various operations in the API, the API description provided by Microsoft is crawled by a crawler, the description is supplemented and enriched by using a data enhancer, and the sentence embedding of the API description is extracted by using a deep learning model as the API description feature. The two parts of the features are combined as API intrinsic semantic features to improve the effect of malware detection.

- We conduct extensive experiments on public datasets and verify the excellent performance of this method in malware detection.

The rest of this article will be organized as follows: Section 2 introduces related work on API-based malware detection and LLM-based data enhancement; Section 3 presents our malware detection method; Section 4 gives a comparison of the experimental results of our experimental setup and other methods. Ablation studies and analysis are discussed in Section 5. The last section summarizes the content of the full text.

2. Related Work

In this section, the related work on LLM-based data enhancement and malware detection is presented.

2.1. LLM-based Data Enhancement

The training of LLMs on large-scale network data has endowed them with comprehensive comprehension capabilities, demonstrating superior reasoning and generalization abilities, which have driven innovations in several domains, including creative writing, interactive dialogue systems, and search engine design. There are many scholars working on data annotation, feature engineering, and data augmentation using LLM, empowering other fields and advancing science and technology through LLM's generalization capabilities.

Cao et al. uncover the multifaceted applications of LLMs, ranging from elevating customer support and fortifying fraud detection to reshaping market analysis and prediction [19]. Han et al. proposed the FeatLLM framework for feature engineering using LLM, which analyzes feature-task relationships, develops rules to generate new features, and then reasons about them with simple downstream models [20]. Liu et al. developed the MixSelfConsistencyQueryEngine to augment tabular data using LLM, which aggregates results from textual and symbolic reasoning through a self-consistency mechanism (i.e., majority voting) and achieves state-of-the-art performance [21]. Deng et al. propose a semi-supervised learning approach to augment market opinion analysis with LLM. The LLM is first used to generate weak financial opinion labels for Reddit posts, and then these data are used to train a small model [22]. Lee et al. propose to use off-the-shelf LLMs instead of human-labeled preference data, train reward models based on these labeled data, and fine-tune the LLMs with a reinforcement learning approach to better understand human preferences [23]. While researchers are currently introducing LLM as an alternative to manual analysis by experts in many domains, there is a gap in similar work in the field of cyberspace security, where many antecedent work relies on cyberspace security experts.

2.2. Executables Based Malware Detection

Methods based on executables typically utilize machine learning or deep learning methods to extract patterns and features of malicious software from a large number of executables, achieving high-precision malicious software detection. For example, Hemalatha et al. propose an efficient malware detection system based on deep learning which uses the reweighted class-balanced loss function in the final classification layer of the DenseNet model to achieve significant performance improvements in classifying malware by handling imbalanced data issues [12]. Shaukat et al. first visualize portable executable files as images, then use a fine-tuned deep learning model to extract deep features from those images. Finally, they employ SVM for malware detection based on these deep features [13]. Rajasekhar et al. propose a Deep Learning-based Bidirectional-Gated Recurrent Unit-Convolutional Neural Network (Bi-GRU-CNN) model for detecting and classifying internet of thing (IoT) malware using ELF binary file byte sequences as input features [24]. Saleh et al. introduce a high-performance malware detection system that combines deep learning and feature selection methodologies to differentiate malware from benign traffic [25]. However, due to the rich information embedded in executables and their large file size, the time cost of detecting executable files is relatively high.

2.3. API Based Malware Detection

Methods based on API mostly focus more on whether the API is called and its frequency. They often overlook the impact of API parameters on API and usually employ manual analysis for the relationships between API. For example, Singh et al. captured software API calls using the Cuckoo sandbox, selected multiple types of API, and used their invocation as features for optimizing machine learning algorithms to detect malicious software [7]. Amer et al. proposed an Android malware detection technique based on API and permissions, collecting application features by obtaining the most frequently used API calls and permissions and utilizing machine learning algorithms

for malware detection [8]. Amer et al. employed word2vec to extract contextual relationships between API sequences, cluster similar API, and ultimately detect malware based on a Markov chain [9]. Sharma et al. extracted important features for malware detection from API call sequences, invocation situations, and called frequencies obtained from the dynamic analysis of malicious and benign samples. They used the TF-IDF method to determine the importance of each feature in these feature sets and evaluated the feature effectiveness using machine learning algorithms such as decision trees, support vector machines, logistic regression, and k-nearest neighbors [10]. Ndibanje et al. calculated the called frequency of each API in each API sequence, represented each malware sample using this frequency vector, and then applied the KNN machine learning algorithm for feature extraction [11]. While these works recognize the superior performance of API calls in malicious software detection, methods based on machine learning models often struggle to adequately consider the internal dependencies within sequences.

With the introduction of deep learning-related technologies, researchers have begun to explore malware detection from the perspective of API call sequences, utilizing deep learning models to model sequence data and uncover dependencies within the data. For example, Liu et al. analyze malware and normal software samples using the sandbox software. They collect API calls and remove duplicate API calls. Then they use vectorization techniques from natural language processing (NLP) to explore relationships between the API and obtain vectors representing API calls. Finally, they employ Bi-directional Long Short-Term Memory (Bi-LSTM) for malware detection [14]. Maniriho et al. utilize the tokenizer in the Keras framework for tokenizing and encoding API. They propose an automatic feature extraction approach based on Convolutional Neural Networks (CNN) and Bi-directional Gate Recurrent Unit (BiGRU) deep learning architecture [15]. Feng et al. propose a novel documentation-augmented Windows malware detection framework to extract the information of official Windows API documentation and construct API graphs [26]. The above studies have overlooked the impact of API parameters on malware detection.

Recently some researchers have started to explore malware detection from API sequences with parameters. For example, Zhang et al. used a hash method to extract heterogeneous features from API names and run-time parameters. These features were further concatenated and input into a deep learning model that aggregates multiple gated CNN models and bidirectional LSTM [16]. Li et al. proposed a hybrid feature encoder for extracting semantic features from API names and parameters. Subsequently, an API call graph was derived from the API call sequence, transforming the relationships between API calls into structural information of the graph. Finally, a graph neural network was designed for malicious software detection and type classification [17]. Chen et al. introduced a classification method based on rules and clustering to evaluate the sensitivity of parameters to malicious behavior, obtaining a parameter-enhanced API call sequence. Based on this sequence, native embeddings and classification label embeddings were applied to API calls, connecting the two to represent the API. The embedded sequence was then input into a deep neural network to train a binary classifier for malicious software detection [18]. Zhao et al. employ parameter-augmented semantic chains to improve the system's resilience to unknown parameters and design a deep learning model consisting of gated CNN, Bi-LSTM, and an attention mechanism to extract semantic features embedded within the API sequences and improve the overall detection accuracy [27]. Zhou et al. apply a Multi-Layer Perceptron to analyze the API argument features and propose a hybrid model that combines the Hierarchical Attention Network and Multi-Layer Perceptron to detect malware [28].

3. Method

In this section, the proposed method and the motivation behind it are described in detail. The system framework is shown in Figure 1, API Semantic Feature Encoder, API Co-occurrence Feature Encoder, DNN.

The API encoder can be divided into two parts. One is the API semantic feature encoder, the other is the API co-occurrence feature encoder. The API semantic feature encoder extracts intrinsic semantic features from API names and Microsoft's official API definitions based on the LLM's prompt engineering and sentence embedding techniques. The API co-occurrence feature encoder mines the contextual co-occurrence features of API from API call sequences based on the word2vec.

During the TCN-GRU model, the API embedding sequence is input into our designed deep neural network. The TCN layer is employed to extract overall temporal features from the API call sequence. Subsequently, the GRU layer is utilized for a comprehensive analysis of the associated features extracted by the TCN layer. Finally, the associated features extracted by the GRU layer are input into the linear layer, and the sigmoid activation function is applied for classification, determining whether the corresponding software for the API call sequence is from malware.

3.1. API semantic feature encoder

The API semantic feature encoder consists of three parts as shown in the figure: the LLM-based API data

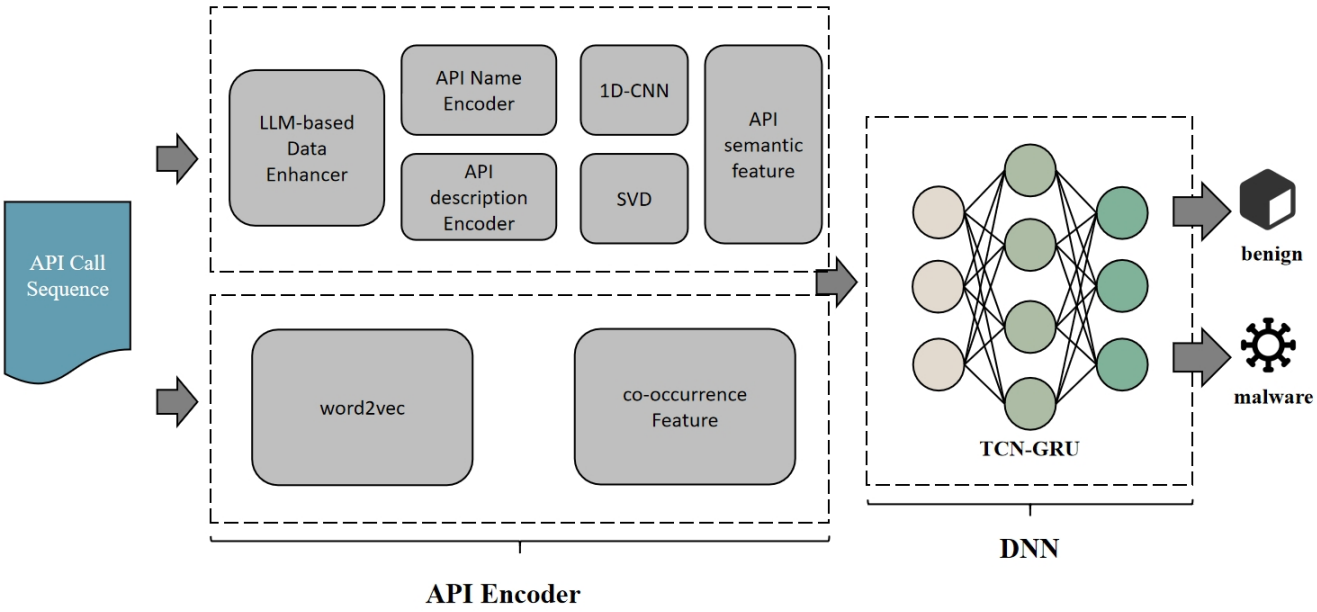

Figure 1. Overview of Our Malware Detection Method. The system mainly consists of two parts. (1) API Encoder: to extract the semantic feature of API. (2) DNN: to build the model that models the dependency of API, then train this model to classify.

enhancer, the API name-enhanced feature encoder, and the API description-enhanced feature encoder.

Currently, many research works tend to regard API as a meaningless symbol when utilizing API call sequences for malware detection and tend to study the dependency relationship between symbols and symbols, i.e., between different API. Some researchers statistically analyze API call sequences, while others conduct NLP-like studies on API calls.

As a matter of fact, API is not just a meaningless symbol, but we can dig rich deep information from the names and meanings of API. The naming of API names usually follows certain rules, and the names can be used to intuitively understand the scope of the API's functions and uses. For example, API "CreateFile" is responsible for creating files, API "ReadProcessMemory" is responsible for reading the memory of a specified process, API " VirtualAlloc" is responsible for reading the memory of a specified process, and API "VirtualAlloc" is responsible for reading the memory of a specified process. VirtualAlloc" is responsible for allocating virtual memory, etc. The definition of API is very important in development, which can help developers quickly understand the use of API, and the definition usually contains a description of the function of the API, the use of scenarios, and precautions, to avoid the developer misuse or abuse of API. CreateFile" is to create or open files, devices, pipelines, etc., allowing the specification of file access rights, sharing modes, and creation methods; API 'GetWindowText' is to retrieve the text of the title bar of the specified window, which applies to non-control windows. The API "GetWindowText" retrieves the title bar text of the specified window, which applies to non-control windows. This information can enrich the semantic information, solve the problem of long-tailed vocabulary, and then improve the representation of API vectors to achieve the effect of enhancing the API representation capability.

Therefore, we would like to extract information about operations, operation objects, etc. from the API names. Also, it is desirable to get a paraphrase about what kind of work each API is responsible for. The design of each component is as follows:

- LLM-based API Data Enhancer: Extracting information such as operator, operation, operation object, etc. from API names and providing a description for the API is a challenging task.API names are usually short and contain multiple levels of semantic information, e.g. "CreateFile" implies that the operator is the system/user, the operation is creation, the operation object is the file, etc. However, this information can be vague and inconsistent. API names are often short and contain multiple levels of semantic information, e.g., "CreateFile" implies that the operator is the system/user, the operation is to create, the object of the operation is a file, etc., but this information can be ambiguous and inconsistent. Precise annotation of API names requires a deep

Table 1. Detail of Prompt from LLM-based API Data Enhancer

Prompt Type	Prompt
API Name Extraction	You are an expert in the field of cyberspace security and Windows programming, familiar with various Windows API, and have many years of experience in using Windows API. Please extract for me the operations, operation categories, operation objects, operation object categories, and API categories from some of the specified API, which is “Networking”, “Persistence”, “Encryption”, “Anti-Analysis”, “Stealth”, “Execution”, “Miscellaneous”. Miscellaneous”. Here are some examples: “API name”: “CreateFile”, “action”: “create”, “action category”: “add”, “object”: “file”, “object category”: “file”, “API category”: ” persistence". Here are the API that needs to work:
API Description Generation	You are an expert in the field of cyberspace security and Windows programming, familiar with various Windows API, and have many years of experience in using Windows API. Please provide a paraphrase for these API as per the official Microsoft API paraphrases. Here are some examples:“API name”: “CreateFile”, “description”: “Creates or opens files, devices, pipes, and so on, allowing file access permissions, sharing modes, and creation methods to be specified.”. Here is the API that needs to work:

understanding of naming conventions and contextual semantics. In the past, when faced with similar tasks, it relied on manual analysis by experienced senior experts. However, this may lead to low efficiency, and consistency is difficult and costly to ensure when multiple experts work together. LLM, by pre-training on a large amount of code and natural language corpus, can efficiently understand and extract multi-level information from API names, and assist in generating descriptions for API. To achieve this goal, a good prompt must be designed. Therefore, the LLM-based data enhancer is designed to guide LLM through prompt engineering in a few-shot manner to accomplish these two tasks: extracting and inferring structured information from API names, and generating explanations for API. The prompt used is shown in the table 1.

- API Name-enhanced Feature Encoder: With the API Data Enhancer, the operation, operation category, operation object, operation object category, and API classification are extracted from the API name. For example, the operation of API “CreateFile” is “Create”, the operation classification is “Add”, the operation object is “File”, and the object of the operation is “Add”. “File”, the operation object is classified as ‘File’, and the API classification is ‘Persistence’. The API name augmentation feature encoder will be utilized to extract the intrinsic semantic features embedded in API names. Specifically, each API name is converted into a 5-tuple (operation, operation classification, operation object, operation object classification, API classification) to form a name semantic chain. Then, an embedding layer is used to convert the name semantic chain into a matrix. Finally, the one-dimensional convolutional layer is applied to extract the features of the name semantic chain to obtain the API name features.
- API Description-enhanced Feature Encoder: With the API Data Enhancer, we are able to get a detailed description of each API feature. For example, the definition of API “CreateFile” is to create or open a file, device, pipeline, etc., which allows us to specify the file access rights, sharing mode, and creation method. We will utilize the API description augmentation feature encoder to extract the intrinsic semantic features embedded in the API description. Specifically, for each API paraphrased text, we will use a pre-trained sentence embedding model to dig deeper into the semantic information of the text and convert it into a fixed-length vector representation. At the same time, in order to avoid dimensionality catastrophe, we will also use Truncated SVD to downscale the sentence embeddings.

3.2. API co-occurrence feature encoder

In fact, many API is not isolated from each other, they depend on each other to help users realize specific functions. For example, CreateFile for opening a file often appears together with file manipulation API such as ReadFile for reading a file or WriteFile for writing a file to form a context for file manipulation. Similarly, in network communication, WSAStartup, connect, send, and recv often appear together to form a context about

network communication, reflecting the lifecycle of a network session.

These co-occurrence patterns are particularly useful in malware analysis because malicious code often uses specific API sequences to accomplish malicious tasks, such as obtaining privileges, loading malicious modules, and performing process injection. For example, the co-occurrence of VirtualAllocEx and WriteProcessMemory is often a sign of process injection. By identifying the co-occurrence patterns of API, malicious behavioral patterns can be more effectively detected and help distinguish between regular software and malicious programs.

Word2vec is a natural language processing technique that represents words as low-dimensional vectors and generates vector representations by capturing co-occurrence relationships between words. The basic idea is that two words should be close to each other in the vector space if they frequently co-occur in similar contexts. Word2vec consists of two main models, the Skip-Gram model which predicts the context based on the central word, and the CBOW model which predicts the central word based on the context and is trained by an objective function that maximizes the prediction probability. In this way, the models can embed semantically similar words in regions of vector space close to each other, allowing words with high co-occurrence frequency to form clusters in vector space.

The API co-occurrence feature encoder utilizes the Word2Vec technique for feature encoding. Specifically, each API is considered as a "word" and each software sample as a "document", and contextual features are generated based on the co-occurrence of API. By using the CBOW model, the encoder can predict the central API based on the contextual API and train with an objective function that maximizes the probability of predicting the central word. For example, if two API functions frequently appear in the same context, the CBOW model will utilize this co-occurrence information to predict the target API by the surrounding context, so as to learn the relationship between the context API and the target API and construct co-occurrence vectors, so that API with similar functions or uses are close to each other in the vector space. The target function formula is shown below equation 1:

$$\tau = \sum_{w \in C} \log p(w\,(context\,(w)) \tag{1}$$

Where w denotes any word in the corpus C.

3.3. DNN

Previous research has often treated the problem of malware detection based on API call sequences as a sequence classification problem, typically employing deep learning models such as Recurrent Neural Networks (RNNs) and LSTMs. Models of this kind [29, 30] determine each output based on both the current input and previous information. Therefore, they can handle sequence data, uncover temporal information in the data, and capture dependencies among sequence data. However, when dealing with long sequences, these models may encounter issues such as vanishing or exploding gradients as the network depth increases.

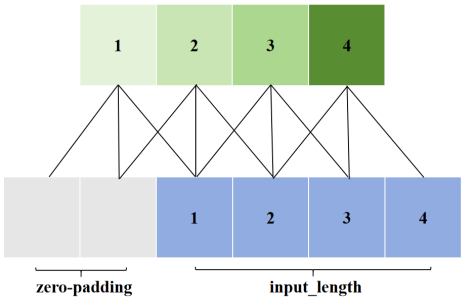

Figure 2. Causal Convolution in TCN

TCN is a type of deep neural network model based on one-dimensional convolution [31]. Due to its ability to parallelly compute data from all time steps, as well as its powerful capability to model long-term dependencies with fewer parameters, TCN has been widely applied in areas such as speech recognition, motion detection, and time series classification. It is built on two mechanisms: causal convolution and dilated convolution.

Causal convolution is illustrated in Figure 2. The value at position t in the output layer depends only on the values at position t and earlier in the input layer. Unlike traditional convolutional neural networks, causal convolution cannot see future data; it has a unidirectional structure. In other words, there must be a cause before there is an effect, making it a strictly time-constrained model, hence the name causal convolution.

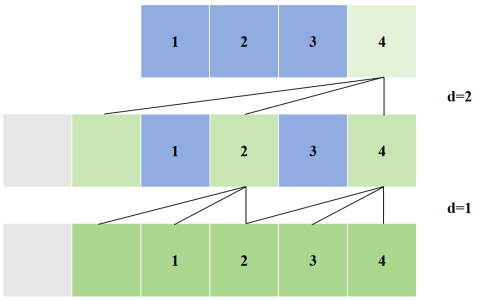

Figure 3. Dilated Convolution in TCN

Simple causal convolution still has the problem of traditional CNN, that is, the length of modeling time

is limited by the size of the convolution kernel, and it is difficult to obtain longer dependencies. The solution to this problem is dilated convolution, as shown in Figure 3. Dilated convolution allows for interval sampling of the input during convolution, and the sampling rate is controlled by the dilated coefficient, i.e. d in the figure. $d = 1$ means that every point in the input process is sampled, and $d = 2$ means that every two points in the input process are sampled once as input. Generally speaking, the higher the level, the greater the value of d. Therefore, dilated convolution makes the size of the effective window increase exponentially with the number of layers. In this way, the convolution network can use fewer layers to obtain a large range of receptive fields.

To ensure that the receptive field of the TCN covers the entire history, i.e. the complete sequence, it is necessary to control the number of layers to achieve a wide receptive field.

For the given sequence length denoted as l, kernel size denoted as k, dilation base denoted as b, and number of layers denoted as n, the following inequality equation 2 needs to be satisfied to cover the complete history:

$$1 + (k-1) \cdot \frac{b^n - 1}{b-1} \geq l. \tag{2}$$

Solving for n to obtain the minimum required number of layers as following equation 3:

$$n = \left\lceil \log_b \left(\frac{(l-1) \cdot (b-1)}{(k-1)} + 1 \right) \right\rceil. \tag{3}$$

The two mechanisms of causal convolution and expansion convolution are used to make the output of each moment in the TCN network only convoluted with the input at that moment and before. Therefore, the output of TCN maintains a temporal sequence. In other words, while TCN employs convolutional operations to extract features from the sequence, these features still possess temporal order. Consequently, TCN can be combined with neural networks such as RNN, LSTM, and GRU.

RNN is designed to handle sequential data and capture temporal dependencies within the input sequences. However, RNNs suffer from vanishing and exploding gradient problems, making it difficult to capture long-range dependencies. LSTM addresses this issue by introducing gating mechanisms (input gate, forget gate, and output gate) and memory cells, allowing them to retain important information over longer time horizons. GRU, a simplified version of LSTM, contains only an update gate and a reset gate, resulting in fewer parameters and more efficient computation. GRU [32] typically offers performance comparable to or even better than LSTM, especially in scenarios where model complexity and computational resources are limited. Therefore, by combining TCN with GRU, we construct the deep neural network shown in Figure 4. The deep neural network is used to determine whether the input API call sequence is from malware. The model consists of the TCN layer, the GRU layer, and the Linear layer.

The model will take the API call sequence as input and determine whether the API call sequence originates from malware. The specific process is as follows: first, based on the trained API embedding from the previous step in API Embedding, construct the embedding layer to transform the input API call sequence into the API embedding sequence. Then, using the TCN layer, explore the dependency relationships between API calls, extracting temporal features of the API call sequence. Next, input the feature vectors output by the TCN layer into the GRU layer, overlaying bidirectional feature information. This further explores the intrinsic sequential correlations of the API call sequence in both forward and backward directions, extracting deeper temporal features. Finally, input the feature vectors extracted from the GRU layer into the fully connected layer. Through the sigmoid activation function, classify whether the input API call sequence is from malware.

Additionally, the model applies Adam as the optimizer and supervises each input sequence with the label. To measure the loss of the training stage, the binary cross-entropy function is used as follow:

$$BCE = -(y \log(p) + (1-y) \log(1-p)), \tag{4}$$

where y is binary label and p is the probability of y.

4. Experiment

This section focuses on a detailed description of the dataset used for the experiment, the environment in which the experiment was conducted, and the results of the experiment.

4.1. Experiment Setting

Four publicly available datasets are used to evaluate our approach. The first dataset [33] contains a total of 8087 samples divided into three batches, each batch divided by year. Each sample consists of a pair of API call sequences and corresponding labels. The API call data is collected via the CAPE sandbox and the labels are generated by VirusTotal. We combined the data from all years; the second dataset [34] contains 42,797 malware API call sequences and 1,079 normal software API call sequences. Each API call sequence consists of the first 100 non-repeating consecutive API calls related to the parent process, which are derived from the Cuckoo Sandbox report; the third dataset [35] is generated by Cuckoo Sandbox and is based on the analysis of

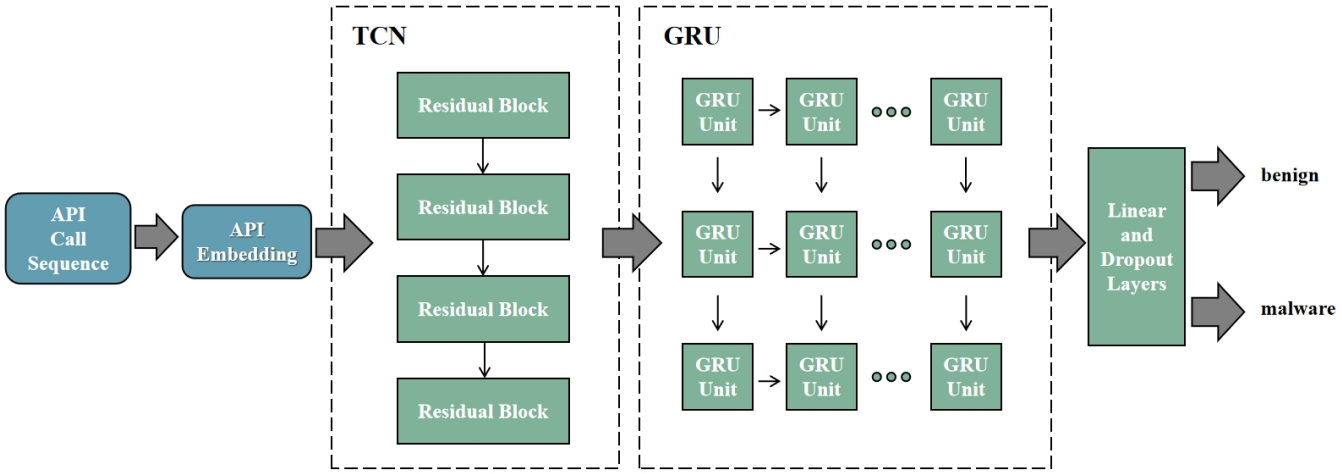

Figure 4. TCN-GRU Model Architecture. The TCN-GRU model consists of four parts. (1) Embedding: to embed API call sequence. (2) TCN: to explore the dependency relationships between API calls. (3) GRU: to extract temporal features of the API call sequence. (4) Linear: to classify where the API sequence is from.

API calls for the Windows operating system, which is intended to provide cybersecurity researchers with malware analytics data suitable for machine learning applications. The dataset contains 5727 malicious samples and 1380 sub-malicious samples. Due to the significant imbalance in the number of positive and negative samples in the above three datasets, an oversampling technique is used to augment the minority class samples in the training set so that the ratio of positive and negative samples is adjusted to at least 7:3, ensuring that the model is able to fully learn the features of the minority class samples during the training process and improve the classification performance. Dataset 4 [36] was generated with a sandbox and was used as the dataset for the Malicious Code Detection track of the DataCon 2019 Big Data Security Analytics Competition. The dataset contains a total of 60,000 software samples, of which 40,000 are malware and the other 20,000 are benign.

During the training, the five-fold cross-validation method is employed, where the training set is randomly divided into five equally sized subsets. Four subsets are used for training, and the remaining one is used for validation in each iteration. This process is repeated five times, and the average results are obtained. Simultaneously, the results on the training set are referred to as training results, the results on the validation subset as validation results, and the results on the test set as test results.

In the proposed model, considering the median and mean of the API sequence lengths in the dataset, we have constrained the length of the API sequences to 1000. Simultaneously, we set the size of the API embeddings to 50 based on the size of the corpus and the cost of unsupervised training of word2vec. Considering the complexity of the malware detection task and the length of the API call sequence, we also set the kernel size of the TCN model to 3 and the output channel of the TCN model to [100, 100, 100, 100, 100, 100, 100, 100, 100, 100]. We set the hidden layer size of the GRU model to 64 to balance task complexity and training consumption. Finally, we set the dropout rate to 0.5 to avoid overfitting.

To prevent overfitting, we employ regularization techniques such as dropout. Additionally, we verify the model with a separate validation set and perform cross-validation to ensure the robustness of our results.

Accuracy, Precision, Recall, and F1-score are used as the evaluation metrics of the proposed method:

$$Accuracy = \frac{|TP| + |TN|}{|TP| + |TN| + |FP| + |FN|}, \quad (5)$$

$$Precision = \frac{|TP|}{|TP| + |FP|}, \quad (6)$$

$$Recall = \frac{|TP|}{|TP| + |FN|}, \quad (7)$$

$$F1 - score = 2 \times \frac{Precision \times Recall}{Presion + Recall}, \quad (8)$$

where TP represents the number of traces that are correctly predicted as malware, TN denotes the number of traces that are correctly classified as benign, FN denotes the number of traces that are malware but are incorrectly predicted as benign, and FP indicates the number of traces that are benign but are predicted as malware.

4.2. Comparison with State-of-the-art

In order to evaluate our approach, we also compared the proposed method with some of the methods proposed by other researchers.

Sharma et al [10] analyzed the API call information extracted from malware and normal samples by dynamically analyzing them as important features for malware detection. Three different feature sets were used in the study; (i) API call usage, (ii) API call frequency, and (iii) API call sequence. Also, these three feature sets were combined into an integrated API call feature set. Subsequently, the TF-IDF method is used to determine the importance of each feature in the feature set and these feature sets are represented as TF-IDF vectors. The effectiveness of malware detection is improved by such enhanced feature sets.

Maniriho et al [15] proposed API-MalDetect for detecting malware attacks in Windows systems. The framework uses an encoder with NLP approach to process API calls and combines a hybrid automatic feature extractor built with CNN and BiGRU to extract features from raw and long sequences of API calls. The proposed framework aims to detect unknown malware attacks and avoid performance degradation over time or due to different malware exposure rates by reducing temporal bias and spatial bias during training and testing. The method automatically identifies unique patterns in API call sequences, effectively distinguishes malware attacks from normal behavior, and provides cybersecurity experts with the most important API call information.

We will conduct experiments in Dataset 1, Dataset 2, Dataset 3, and Dataset 4, and will divide each dataset into two parts, a training set and a test set, in the training set the proportion of normal software samples will be increased by over-sampling techniques, and at the same time the validation set will be sampled in the training set. Four main metrics are considered for experimental evaluation including Accuracy, Precision, Recall, and F1-score.

As shown in Tables 2, 3, 4, and 5, the performance of the proposed model outperforms almost all the models proposed in other studies in Accuracy, Precision, Recall, and F1-score. Our proposed malware detection method achieves superior performance compared to the methods of Sharma et al. and Maniriho et al. Our proposed method achieves an F1-Score of 92.77% for the test results in Dataset 1, which is 27.23% and 31.29% better than Sharma et al. and Maniriho et al.'s methods, and 88.14% for the test results in Dataset 2, which is 38.76% better than Maniriho et al. and 38.76% better than the Sharma et al.'s method by 8.67%; the F1-Score of the test results in Dataset 3 achieved 80.83%, which exceeded the methods of Sharma et al. and Maniriho et al. by 3.48% and 22.86%. The F1-Score of the test results in Dataset 4 is as high as 98.30%, exceeding the methods of Sharma et al. and Maniriho et al. by 3.04% and 5.63%. These results show that our method not only considers the impact of API intrinsic semantics on malware detection, but also enhances the method's ability to characterize API through feature extraction techniques from LLM and Microsoft official documents, which further improves the accuracy of malware detection, and the experimental results verify the effectiveness of these treatments, and thus the performance in the experiments is able to outperform them.

From the experimental results, it can also be found that our proposed method performs better in Dataset 1 and Dataset 3, while it performs poorly in Dataset 2. The reason may be that actually the API call sequences in Dataset 1 and Dataset 3 are complete from the sandbox software, while the API call sequences in Dataset 2 are de-duplicated. The API in Dataset 2 can only appear once in each call sequence, so the internal dependencies between their sequence-to-API calls are corrupted, leading to the frustration of deep learning's long sequence analysis methods, while machine learning and statistical analysis methods work better.

4.3. Comparison with Different Model

We compare the proposed method with some popular deep learning methods nowadays (including RNN, TCN, TCN-LSTM) in terms of F1-score, Precision, Recall, and Accuracy. This is done by training validation on the training set of Dataset 4 and testing on the test set of Dataset 4. We will input the feature-processed API sequences into all these baseline models separately.

The experimental results are shown in Tables 6, 7, where our proposed method outperforms the baseline model in both the validation and test sets. For example, in the test results, our proposed model is 98.30%, 98.24%, 98.36%, 98.54% on F1-score, Precision, Recall, and Accuracy, which is 1.15%, 1.79%, 0.38%, and 1.01% better than the best baseline model. The improvement mainly comes from the fact that our proposed method considers the significance of API intrinsic semantics in malware detection, using LLM to play the role of a security expert with the help of Microsoft's official API description, extracting deeper information from the API and extracting features from it using feature extraction techniques. In terms of modeling, comparing the temporal neural network model, i.e., RNN, our model utilizes the two mechanisms of causal convolution and inflationary convolution of spatiotemporal convolutional neural networks to obtain a longer sensory field and extracts the features of the longer context in the sequence of software API calls. While comparing the TCN model, our model utilizes

Table 2. Experiment Result in Dataset1

Method	Validation Results (%)				Test Results (%)			
	F1-score	Precision	Recall	Accuracy	F1-score	Precision	Recall	Accuracy
API-MalDetect [15]	65.89	83.20	64.96	76.71	61.48	59.12	65.77	97.38
TFIDF [10]	93.01	93.29	92.75	93.75	65.54	60.24	91.02	95.65
Proposed Method	**99.06**	**99.15**	**98.97**	**99.16**	**92.77**	**88.32**	**98.47**	**99.36**

Table 3. Experiment Result in Dataset2

Method	Validation Results (%)				Test Results (%)			
	F1-score	Precision	Recall	Accuracy	F1-score	Precision	Recall	Accuracy
API-MalDetect [15]	39.30	32.37	50.00	64.75	49.38	48.77	50.00	97.54
TFIDF [10]	97.09	97.59	96.59	96.00	**96.81**	**97.49**	96.14	95.62
Proposed Method	**97.87**	**97.93**	**97.82**	**98.07**	88.14	82	**97.56**	**98.58**

Table 4. Experiment Result in Dataset3

Method	Validation Results (%)				Test Results (%)			
	F1-score	Precision	Recall	Accuracy	F1-score	Precision	Recall	Accuracy
API-MalDetect [15]	52.94	61.1	56.45	59.52	57.97	59.34	57.35	76.39
TFIDF [10]	77.56	82.36	77.48	78.80	77.35	**80.05**	75.42	86.89
Proposed Method	**83.09**	**83.54**	**82.98**	**83.24**	**80.83**	78.95	**83.48**	**87.03**

Table 5. Experiment Result in Dataset4

Method	Validation Results (%)				Test Results (%)			
	F1-score	Precision	Recall	Accuracy	F1-score	Precision	Recall	Accuracy
API-MalDetect [15]	92.65	92.60	92.71	93.71	92.67	92.46	92.89	93.71
TFIDF [10]	95.65	95.50	95.81	96.27	95.26	95.03	95.50	95.92
Proposed Method	**98.7**	**98.62**	**98.78**	**98.89**	**98.3**	**98.24**	**98.36**	**98.54**

Table 6. Validation Results (%) of Comparison with Deep Learning Model on Malware Detection

Approach	F1-score	Precision	Recall	Accuracy
RNN	89.38	88.64	90.29	90.78
TCN	97.31	96.67	98.05	97.66
TCN-LSTM	97.13	96.45	97.91	97.53
Proposed Method	**98.70**	**98.62**	**98.78**	**98.89**

Table 7. Test Results (%) of Comparison with Deep Learning Model on Malware Detection

Approach	F1-score	Precision	Recall	Accuracy
RNN	88.69	88.05	89.44	90.15
TCN	97.14	96.45	97.93	97.51
TCN-LSTM	97.15	96.44	97.98	97.53
Proposed Method	**98.30**	**98.24**	**98.36**	**98.54**

the gated loop unit to comprehensively analyze the correlation features extracted by the spatio-temporal convolutional neural network on top of the features extracted by the spatio-temporal convolutional neural network and learns the deep temporal dependencies embedded in the data. Finally comparing the TCN-LSTM model, our model relies on GRUs with their simplified gating structure and more efficient training

process and shows a superior performance in malware detection.

5. Discussion

In this section, the effects of the improvements made in the proposed approach are discussed through the ablation study first. Then, the impact of varying the method of dimension reduction performance is discussed.

5.1. Ablation Study

In this study, the ablation experiments aim to evaluate the impact of the improvements made to the methodology in this chapter on the malware detection performance, verifying the contribution of the individual modules to the overall effectiveness of the model. By gradually removing or replacing certain key parts of the model, the importance of each component can be clarified, which in turn helps to understand the strengths and limitations of the model.

The improvements of the methodology mainly include the following parts: (i) API Name-Enhanced Feature Encoder, which extracts the intrinsic semantic features embedded in API names. Specifically, each API name is converted into a 5-tuple (operation, operation category, operation object, operation object category, API category) to form a name semantic chain. Then, an embedding layer is used to convert the name semantic chain into a matrix. Finally, a one-dimensional convolutional layer is applied to extract the features of the name semantic chain to obtain the API name features. (ii) API Description-Enhanced Feature Encoder to extract the intrinsic semantic features embedded in API description. Specifically, for each API description text, a pre-trained sentence embedding model will be used to dig deeper into the semantic information of the text and convert it into a fixed-length vector representation. Meanwhile, in order to avoid dimensionality catastrophe, Truncated SVD will also be used to downscale the sentence embeddings. (iii) API co-occurrence features.

In order to verify the effectiveness of these improvements, we design the following ablation experiments: (1) only name-enhanced feature sets (only-name) (2) API intrinsic semantic feature sets (only-semantic) (3) only co-occurring features are retained (only-sequence) and (4) the complete method. These ablation experiments allow us to comprehensively assess the contribution of each module to the performance of the malware detection system and provide a basis for subsequent optimization.

The results of the ablation experiments in table 8 show that the performance of the model decreases again when only co-occurring features are retained, indicating that the co-occurring features are the key factors in marking the behavioral patterns of malware. The F1-Score, precision, accuracy, and recall of the model drop when only name-enhanced features are included, suggesting that the name semantic information of API calls plays an important role in malware detection. The accuracy and recall of the model rebound in the results of the API intrinsic semantic feature set, suggesting that the paraphrased semantic information of API calls is pivotal to the performance of the model. The best performance was obtained for the complete model, demonstrating the synergy of the individual modules.

In summary, the ablation experiments verify the importance of API intrinsic semantic features and API co-occurrence features in malware detection and provide a strong basis for further optimization of the model.

5.2. Different Methods of Dimensionality Reduction

The performance of the proposed model is mainly affected by the Dimensionality Reduction methods used in the API description features. In this section, we will examine the impact of these factors by configuring the proposed methods using different settings.

In this study, in order to evaluate the impact of different dimensionality reduction methods on the performance of malware detection models, we designed dimensionality reduction experiments based on Truncated SVD, t-SNE, and PCA. Dimensionality reduction techniques are commonly used to reduce the dimensionality of data, improve computational efficiency, and reduce the risk of overfitting. However, dimensionality reduction not only affects the computational performance of the model but may also have a significant impact on the final detection accuracy. These experiments allow us to comprehensively evaluate the effectiveness of dimensionality reduction methods in malware detection tasks.

From the experimental results in table 9, the F1-Score performance of using Truncated SVD, t-SNE, and PCA with dimensionality reduction and without dimensionality reduction in the experiments are 98.30%, 97.39%, 97.87%, and 97.04%, respectively. Without dimensionality reduction, the model performs mediocrely in all indicators, mainly due to the presence of a large amount of redundant information and noise in the high-dimensional data, which affects the model's learning of key features. After dimensionality reduction, the model performance is significantly improved. Truncated SVD shows the best overall results in the malware detection task, which is able to effectively retain the key features of the data while reducing the computational complexity and training time. t-SNE, despite its excellent performance in data visualization, its nonlinear dimensionality reduction

Table 8. Experiment Result in Ablation Study

Performance	Validation Results (%)				Test Results (%)			
	Only-name	Only-semantic	Only-sequence	Complete model	Only-name	Only-semantic	Only-sequence	Complete model
F1-score	97.18	98.19	96.52	**98.70**	96.73	97.83	96.37	**98.30**
Precision	96.50	97.72	95.59	**98.62**	96.02	97.37	95.49	**98.24**
Recall	97.95	98.69	97.65	**98.78**	97.55	98.34	97.44	**98.36**
Accuracy	97.56	98.44	97.00	**98.89**	97.16	98.13	96.83	**98.54**

Table 9. Effect of Different Methods of Dimensionality Reduction

Method	Validation Results (%)				Test Results (%)			
	F1-score	Precision	Recall	Accuracy	F1-score	Precision	Recall	Accuracy
unprocessed	97.12	96.32	98.05	97.49	97.04	96.26	97.96	97.42
t-SNE	97.48	97.57	97.4	97.85	97.39	97.53	97.26	97.78
PCA	98.32	98.11	98.54	98.56	97.87	97.72	98.02	98.17
SVD	**98.70**	**98.62**	**98.78**	**98.89**	**98.30**	**98.24**	**98.36**	**98.54**

property makes it less effective than Truncated SVD and PCA in traditional classification tasks. PCA is not as effective in the traditional classification task in the case of data with low dimensionality or strong linear relationships between features, but PCA is not as effective as Truncated SVD in complex feature spaces.

6. Conclusion

In the paper, a new malware detection method is proposed for API intrinsic semantics. The method first designs the API intrinsic semantic feature encoder, which extracts intrinsic semantic features from API names and Microsoft official API paraphrases based on LLM's cue word engineering and sentence embedding techniques. Then the API co-occurrence feature encoder is designed, which mines the contextual co-occurrence features of API from API call sequences based on the word2vec technique. The API intrinsic semantic features and API co-occurrence features are combined and the dependencies between API calls are captured by the TCN-GRU to improve the malware detection performance. The results on several publicly available malware detection datasets show that our approach achieves better performance than other state-of-the-art methods, and in addition, the ablation experiment results demonstrate the important role of semantics in malware detection algorithms.

Acknowledgement. This work was supported by MIIT Project Industrial Internet identifcation resolution system security monitoring and protection (TC220H078), Qing Lan Project in Jiangsu universities, XJTLU RDF-22-01-020. G. Geng is supported by the Pearl River Talents Plan.

References

[1] AV TEST (2023), Malware statistics[eb/ol], https://www.av-test.org/en/statistics/malware/.

[2] ENISA (2023), Enisa threat landscape 2023, https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023.

[3] KARAK, A., KUNAL, K., DARAPANENI, N. and PADURI, A.R. (2024) Implementation of gpt models for text generation in healthcare domain. *EAI Endorsed Transactions on AI and Robotics* **3**(1). doi:10.4108/airo.4082.

[4] YAO, Y., DUAN, J., XU, K., CAI, Y., SUN, Z. and ZHANG, Y. (2024) A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing* : 100211.

[5] OFOEDA, J., BOATENG, R. and EFFAH, J. (2019) Application programming interface (api) research: A review of the past to inform the future. *International Journal of Enterprise Information Systems (IJEIS)* **15**(3): 76–95.

[6] MICROSOFT (2024), Microsoft windows app development documentation, https://learn.microsoft.com/en-us/windows/apps/.

[7] SINGH, J. and SINGH, J. (2022) Assessment of supervised machine learning algorithms using dynamic api calls for malware detection. *International Journal of Computers and Applications* **44**(3): 270–277.

[8] AMER, E., MOHAMED, A., MOHAMED, S.E., ASHAF, M., EHAB, A., SHEREEF, O. and METWAIE, H. (2022) Using machine learning to identify android malware relying

on api calling sequences and permissions. *Journal of Computing and Communication* **1**(1): 38–47.

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

[10] Sharma, P. (2022) Windows malware detection using machine learning and tf-idf enriched api calls information. In *2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA)* (IEEE): 1–6.

[11] Ndibanje, B., Kim, K.H., Kang, Y.J., Kim, H.H., Kim, T.Y. and Lee, H.J. (2019) Cross-method-based analysis and classification of malicious behavior by api calls extraction. *Applied Sciences* **9**(2): 239.

[12] Hemalatha, J., Roseline, S.A., Geetha, S., Kadry, S. and Damaševičius, R. (2021) An efficient densenet-based deep learning model for malware detection. *Entropy* **23**(3). doi:10.3390/e23030344, URL https://www.mdpi.com/1099-4300/23/3/344.

[13] Shaukat, K., Luo, S. and Varadharajan, V. (2023) A novel deep learning-based approach for malware detection. *Engineering Applications of Artificial Intelligence* **122**: 106030.

[14] Liu, Y. and Wang, Y. (2019) A robust malware detection system using deep learning on api calls. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)* (IEEE): 1456–1460.

[15] Maniriho, P., Mahmood, A.N. and Chowdhury, M.J.M. (2023) Api-maldetect: Automated malware detection framework for windows based on api calls and deep learning techniques. *Journal of Network and Computer Applications* **218**: 103704.

[16] Zhang, Z., Qi, P. and Wang, W. (2020) Dynamic malware analysis with feature engineering and feature learning. In *Proceedings of the AAAI conference on artificial intelligence*, **34**: 1210–1217.

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

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

[19] Cao, X., Li, S., Katsikis, V., Khan, A.T., He, H., Liu, Z., Zhang, L. *et al.* (2024) Empowering financial futures: Large language models in the modern financial landscape. *EAI Endorsed Transactions on AI and Robotics* **3**(1). doi:10.4108/airo.6117.

[20] Han, S., Yoon, J., Arik, S.O. and Pfister, T. (2024), Large language models can automatically engineer features for few-shot tabular learning. URL https://arxiv.org/abs/2404.09491. 2404.09491.

[21] Liu, T., Wang, F. and Chen, M. (2023), Rethinking tabular data understanding with large language models. URL https://arxiv.org/abs/2312.16702. 2312.16702.

[22] Deng, X., Bashlovkina, V., Han, F., Baumgartner, S. and Bendersky, M. (2022), What do llms know about financial markets? a case study on reddit market sentiment analysis. URL https://arxiv.org/abs/2212.11311. 2212.11311.

[23] Lee, H., Phatale, S., Mansoor, H., Mesnard, T., Ferret, J., Lu, K., Bishop, C. *et al.* (2024), Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback. URL https://arxiv.org/abs/2309.00267. 2309.00267.

[24] Chaganti, R., Ravi, V. and Pham, T.D. (2022) Deep learning based cross architecture internet of things malware detection and classification. *Computers & Security* **120**: 102779.

[25] Alomari, E.S., Nuiaa, R.R., Alyasseri, Z.A.A., Mohammed, H.J., Sani, N.S., Esa, M.I. and Musawi, B.A. (2023) Malware detection using deep learning and correlation-based feature selection. *Symmetry* **15**(1): 123.

[26] Feng, P., Gai, L., Yang, L., Wang, Q., Li, T., Xi, N. and Ma, J. (2024) Dawngnn: Documentation augmented windows malware detection using graph neural network. *Computers & Security* : 103788.

[27] Zhao, D., Wang, H., Kou, L., Li, Z. and Zhang, J. (2023) Dynamic malware detection using parameter-augmented semantic chain. *Electronics* **12**(24): 4992.

[28] Zhou, B., Huang, H., Xia, J. and Tian, D. (2024) A novel malware detection method based on api embedding and api parameters. *The Journal of Supercomputing* **80**(2): 2748–2766.

[29] Sherstinsky, A. (2020) Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena* **404**: 132306.

[30] Zargar, S. (2021) Introduction to sequence learning models: Rnn, lstm, gru. *Department of Mechanical and Aerospace Engineering, North Carolina State University, Raleigh, North Carolina* **27606**.

[31] Bai, S., Kolter, J.Z. and Koltun, V. (2018) An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* .

[32] Moshayedi, A.J., Roy, A.S., Kolahdooz, A. and Shuxin, Y. (2022) Deep learning application pros and cons over algorithm. *EAI Endorsed Transactions on AI and Robotics* **1**(1): e7.

[33] Carpenter, M. and Luo, C. (2023) Behavioural reports of multi-stage malware. *arXiv preprint arXiv:2301.12800* .

[34] De Oliveira, A.S. and Sassi, R.J. (2023) Behavioral malware detection using deep graph convolutional neural networks. *Authorea Preprints* .

[35] Yazi, A.F., Çatak, F.Ö. and Gül, E. (2019) Classification of methamorphic malware with deep learning (lstm). In *2019 27th signal processing and communications applications conference (SIU)* (IEEE): 1–4.

[36] kericwy1337 (2019), Malicious-code-dataset. [online], https://github.com/kericwy1337.