Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.080632

ARTICLE

# Android Software Malicious Detection Based on Dynamic Network Traffic Mixing API Information and Feature Importance Analysis

**Kang Yang**[1,2], **Lizhi Cai**[1,2,*] **and Jianhua Wu**[1,2]

[1]Shanghai Key Laboratory of Computer Software Testing & Evaluating, Shanghai, China
[2]Shanghai Development Center of Computer Software Technology, Shanghai, China

*Corresponding Author: Lizhi Cai. Email: clz@sscenter.sh.cn

Received: 13 February 2026; Accepted: 31 March 2026; Published: 08 May 2026

**ABSTRACT:** Accurate malware identification and family categorization remain significant challenges in large-scale Android software analysis. Although deep learning has surpassed traditional machine learning in performance, its widespread adoption is hindered by the computational overhead stemming from feature redundancy and the lack of interpretability inherent in its black-box nature. To address these issues, this paper proposes DroidNTA, a DL-based detection model that fuses network traffic and API features. The model first constructs a simplified API Call Graph by extracting the intrinsic structural attributes of applications, and subsequently generates API feature vectors from invocation sequences using a Markov chain algorithm. These are then integrated with dynamic network traffic features to form a final representation vector of the Android instance. To enhance transparency, DroidNTA performs feature contribution analysis by adjusting fusion parameters and employs Shapley values to quantify global feature importance. Experimental results demonstrate that DroidNTA achieves superior performance in both binary and family classification tasks, yielding an accuracy of 99.74% and a gain of over 20%, respectively. We have released our code at https://github.com/joeyyk/DroidNTA.

**KEYWORDS:** Android; malware; deep learning; network traffic

## 1 Introduction

With the rapid development of the Android system, the total number of Android applications has long-term sustainable growth. Among existing mobile devices, the Android operating system [1] is widely used due to its adaptability, usability, and lower cost. According to statcounter statistics [2], the current mobile market share of Android smartphones is as high as 69.4%. However, malware based on Android mobile devices [3] is also growing explosively, and they continue to spread by taking advantage of the efficient communication characteristics of the network. The existence of malware has posed a great threat to the security of private property and user personal information in Android mobile devices. This situation allows a large number of malicious developers to insert the hidden malicious code, thereby generating massively Android malware. Therefore, Android malware detection task has attracted widespread attention from researchers.

At present, the most effective malware detection methods rely primarily on machine learning and deep learning detection methods, which mainly extract malicious features in Android applications through static or dynamic detection methods. The malicious features include the API sequences that cause malicious behavior [4], the permission characteristics of forcibly obtaining user information [5], and the dynamic

acquisition of applications action behavior [6], and so on. These unique characteristics require researchers to mine the huge and complex Android data information.

To effectively extract the potential malicious features of these malware data, the existing machine learning research work [7] tried to carry out simple statistical quantification of features in the application and brought them into the training model for learning and classification. Usually, feature quantification methods only consider a single feature representation and cannot mine the interactive information between sample features. Besides, the feature extraction methods of classic machine learning models are also difficult to mine the inherent structural characteristics of Android applications. Although the training and detection speed of classic machine learning methods are faster, their upper limit of detection accuracy is lower, and the overall prediction performance of the machine learning model is not as good as that of deep learning detection methods. Deep learning models [8,9] are currently the best malware detection methods in terms of performance. These detection models can fit the malicious characteristics of Android malware well, and the prediction accuracy is also the highest in existing work.

In addition, in terms of feature extraction, many existing deep learning detection models use static methods [10] to extract features. Although, this type of model extracts features quickly, the process of feature extraction is easy to be changed through encryption and obfuscation, and thus the static feature extraction method has obvious disadvantages. The dynamic feature model needs to monitor the application in real time, simulate the execution of the default behavior of the application, and convert it into model features. However, this method takes a long time and requires long-term monitoring.

To address the aforementioned limitations in existing hybrid detection models, specifically high feature redundancy, excessive computational overhead in dynamic simulation, and the lack of decision transparency. We propose DroidNTA, a synergistic deep learning framework that integrates static structural intent with dynamic network behaviors. Specifically, DroidNTA tackles the scalability and redundancy gaps by employing a streamlined API family-level abstraction rather than raw call sequences. By constructing a simplified API call graph and utilizing a Markov-chain-based transition matrix, the model effectively compresses the feature space while preserving essential behavioral semantics, thereby accelerating training convergence. To overcome the real-time feasibility constraints of traditional dynamic analysis, we focus exclusively on application network traffic monitoring. This approach captures the fundamental statistical fingerprints of malware communication in a concise and efficient manner, avoiding the overhead of simulating all system-level behaviors. Furthermore, to resolve the black-box nature of prior fusion-based models, DroidNTA incorporates a quantitative interpretability framework. By dynamically adjusting the static-to-dynamic fusion ratios and calculating global Shapley values, we provide a mathematically grounded explanation of feature contributions, offering deeper insights into the model's decision-making logic under complex obfuscation scenarios. In this paper, we make the following contributions:

1. Dual-Modality Framework: We propose DroidNTA, which fuses static API semantics with dynamic network behaviors. Experimental results show it significantly outperforms state-of-the-art baselines in both detection and family classification.
2. High-Precision Classification: DroidNTA demonstrates superior stability and accuracy in fine-grained multi-classification tasks, effectively identifying diverse malware categories and families.
3. Explainable Fusion Analysis: By optimizing fusion parameters $(\alpha, \beta)$ and calculating Shapley values, we quantify feature contributions, revealing that real-time network traffic is a primary driver of model performance.

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 describes the DroidNTA method in detail. Sections 4 and 5 discuss the experiment set up and evaluation. Section 6 presents the conclusion and future work.

## 2 Related Work

Malware detection methods can be divided into the following two types according to the method of feature extraction: (1) Static detection method. (2) Dynamic detection method. (3) Deep learning method.

### *2.1 Static Detection Method*

Static detection methodologies eliminate the need for actual execution of malware samples, relying instead on reverse engineering [11] and file parsing to extract malicious features from binary files, assembly code [12], and program metadata [13]. As the predominant paradigm in early malware detection, these methods primarily focus on features such as file hashes [14] and operational codes [15]. Due to their non-executing nature, they offer significant advantages in detection speed and low resource consumption, making them suitable for large-scale, coarse-grained screening. Among existing studies, the MOSDroid model [16] constructs features using opcode multiset-encoded sequences, effectively mitigating the interference of code obfuscation and enhancing the robustness and accuracy of Android malware detection. Shafiq et al. [17] build the transition probability matrix of the most relevant n-gram byte sequences through markov chains for files with embedded malicious code. The MCSC method [18] decomposes the code based on the file structure, and converts the decomposed malicious code into a grayscale image based on SimHash. Malware detection based on control flow graph [19], which converts the program running process or code logic into a control flow graph. Moreover, this type of detection model represents the paths of all processes that may run during program execution. Currently, API-based modeling [20] is considered the most effective static approach; by reverse-engineering Android applications, researchers extract API invocation sequences to construct high-dimensional feature vectors. However, these traditional static methods predominantly extract discrete features and fail to effectively mine the intrinsic structural characteristics of samples.

### *2.2 Dynamic Detection Method*

Dynamic detection methodologies [21] involve executing malware samples within controlled environments, such as sandbox [22], to capture runtime behavioral characteristics and assess potential malignancy based on execution trajectories. These methods monitor the software in real-time, recording both behavioral patterns and physical feature information during execution. The most prevalent dynamic approach is malware analysis based on dynamic API call sequences [23]. Tang and Qian [24] extracted API call sequences through dynamic analysis. Malware detection based on network traffic [25] refers to analyzing the characteristics of network activity information in network traffic data. Wang et al. [26] proposed the TrafficAV model, which performs multi-level network traffic analysis to collect as many network traffic features as required by the detection model. The core features of these methodologies encompass system API invocations, network traffic [27], file I/O operations, and memory footprints. Malware detection based on network traffic primarily analyzes the characteristics of network activity information in network traffic data, and is the simplest dynamic detection method [28]. Furthermore, it enables the recording of complete attack chains, providing essential support for malware attribution and threat analysis. However, dynamic detection suffers from significant computational overhead, prolonged detection cycles, and vulnerability to anti-environment evasion. Moreover, the delayed execution or conditional triggering mechanisms of certain malware can lead to incomplete behavioral feature extraction.

### 2.3 Deep Learning Method

Deep learning detection methods, characterized by their autonomous feature learning capabilities, have transcended the limitations of manual rule design inherent in traditional approaches, particularly within the domain of malware detection. Cutting-edge solutions based on Graph Neural Networks (GNNs) [29] and Transformer architectures [30] have emerged as the research mainstream, accurately adapting to the structural attributes and behavioral logic of mobile software. Compared to earlier models such as CNNs [31,32], these architectures can deeply mine the semantic correlations of Android applications, significantly enhancing performance against evasion tactics like packing. Regarding structural features, Graph Convolutional Networks (GCNs) [33] and Graph Attention Networks (GATs) [34] leverage their topology-aware advantages to aggregate deep features of nodes and edges by parsing decompiled function call graphs and API dependency graphs [27]. For sequential and global features, the Transformer architecture utilizes self-attention mechanisms to capture intrinsic relationships within long API call sequences and multi-modal heterogeneous features. Malsort model [35] proposes a lightweight and efficient scheme for image-based malware classification, utilizing masked self-supervision combined with Swin Transformer [36] to balance both accuracy and efficiency. Although these methods adapt to malware variants without expert knowledge, they still face challenges such as weak model interpretability problem.

## 3 Proposed Method

### 3.1 Overview

In this section, we introduce the DroidNTA model in detail. The overall architecture of the model is shown in Fig. 1. The model can be divided into four parts: (1) API feature extraction; (2) Network traffic feature extraction; (3) Feature fusion and prediction; (4) Interpretability analyis.

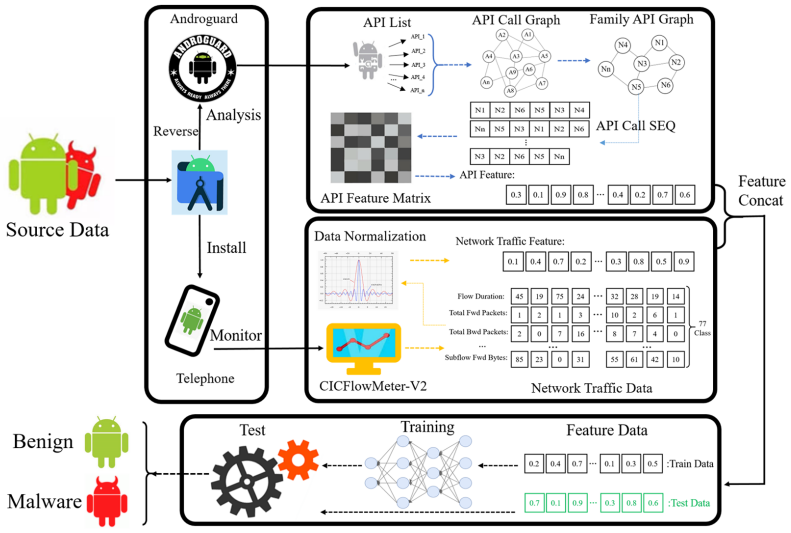



**Figure 1:** The overall architecture of the DroidNTA model.

### 3.2 API Feature Extraction

API features are widely used in Android malware detection, but due to the large number of unique API nodes, it is difficult to input all the data into the model for calculation. To address this issue, existing research only considers the correlation between sensitive malicious API nodes and malware, but this method has obvious limitations [37]. Therefore, we use the API feature extraction method of the MaMaDroid model [38].

First, we use the Androguard (https://androguard.readthedocs.io/en/latest/) tool to reverse compile and obtain the API call graph of the Android application, then abstract the API in the graph into a family node, and build a Family API graph. The process is shown in Fig. 2.

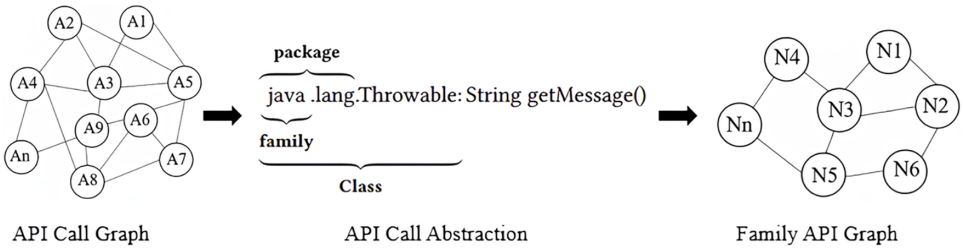



**Figure 2:** Build the family API graph.

After obtaining the Family API graph, we extract the API sequence and combine it with the Markov chain model to calculate the API feature matrix of each Android application. The Markov chains are memoryless models where the probability of transitioning from one state to another depends on the current state [39]. In static Family feature graph, the malware APIs are represented as a set of nodes. Among them, the transition probability $P$ of the node is as shown in the Eq. (1):

$$p_{ij} = P\left(X_{n+1} = j | X_n = i\right) \quad i, j \in S \tag{1}$$

The $X_n$ represents the state of the node. We record the state space as $S = 1, 2, \ldots, m$, which is a finite number of node states in total. Where $p_{ij}$ represents the probability of tansition from node $i$ to node $j$. In addition, the probability of node $X_{n+1}$ only depends on starting a node $X_n$, and there is no connection with previous node information. the calculation process is shown in Eq. (2):

$$p_{ij} = P(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = P(X_{n+1} = j | X_n = i) \tag{2}$$

The transition probability between connected nodes is non-negative and satisfies the equation:

$$\sum_{i}^{S} \sum_{j}^{S} p_{ij} = 1 \tag{3}$$

The final APIs feature vector $API\_Vec$ are constructed by the markov transition matrix $p_{ij}$, the calculation process is shown in Eq. (4):

$$API\_Vec = W * p_{ij} \tag{4}$$

The $W$ is the transformation parameter of the vector dimension, node $i$ and $j$ are Android API.

### *3.3 Network Traffic Feature Extraction*

The network traffic feature come from the dynamic characteristics of Android applications. After installing benign and malicious applications on real Android phones. Researchers uses the data collect tool to monitor these applications in three states. The classification of network traffic feature are shown in Table 1.

**Table 1:** Network traffic feature statistics.

| Category | Feature Type | | | Total |
|---|---|---|---|---|
| Address Information | Flow ID | Source IP | Source Port | 7 |
| | Destination IP | Destination Port | ... | |
| Network Traffic Information | Flow Duration | Total Fwd Packets | Flow Bytes/s | 70 |
| | Flow Packets/s | Flow IAT Mean | ... | |
| Protocol Type | FIN Flag Count | SYN Flag Count | RST Flag Count | 8 |
| | PSH Flag Count | ACK Flag Count | ... | |

In order to facilitate the calculation of the deep learning model, each network traffic feature data is regularized by the following equation:

$$NT\_Vec = \frac{F - F_{min}}{F_{max} - F_{min}} \tag{5}$$

where $F_{min}$ and $F_{max}$ are the minimum and maximum values in the each feature $F$, respectively. Finally, we obtain the network traffic feature vector of each application.

### *3.4 Feature Fusion and Prediction*

After obtaining static API features and dynamic network traffic features, we build the final malware data features through weight parameters $\alpha$ and $\beta$. The parameters fusion process is shown in Eq. (6).

$$x = \alpha \times NT\_Vec + \beta \times API\_Vec \quad (\alpha + \beta = 1) \tag{6}$$

To capture the complex nonlinear correlation between API call sequences and network traffic characteristics, the DroidNTA model incorporates an enhanced residual multilayer perceptron. The core of this architecture is the Residual Block, which mitigates the vanishing gradient problem and enhances feature representation depth. The transformation within a single residual block is defined as follows.

$$h_{out} = \text{LeakyReLU}(\text{BN}(W_{out} \cdot x + b_{out})) + SC(x) \tag{7}$$

$W_{out}$ and $b_{out}$ are the weight matrix and bias vector of the linear layer, respectively. BN denotes Batch Normalization, and LeakyReLU is the activation function, x is the feature sample. The term $\text{SC}(x)$ represents the identity mapping or a linear projection used to align dimensions, ensuring a robust gradient flow during backpropagation. After passing through three hierarchical residual blocks, the high-dimensional latent features are mapped to the label space. For a given input, the model generates a raw output vector (logits) $z$, which is then transformed into a probability distribution $\hat{y}$ using the Softmax function:

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad j \in \{1, 2, \ldots, K\} \tag{8}$$

where $K$ is the number of categories in the dataset. $e^{z_j}$ is the model raw output (logits) for the $j$-th class. When $k = 5$, it is used for family classification (5-Class). When $k = 43$, it is used for category classification (43-Class). The node corresponding to the largest value in the $\widehat{y}$ vector is our model predicted value. In model

train process, the loss function is defined as the cross entropy of the prediction and the ground truth, the calculation process is shown in Eq. (9):

$$Loss(\widehat{y}) = -\sum_{v}^{V}\sum_{k=0}^{K-1} y_v[k] * log(h_t[v][k]) \tag{9}$$

where parameter $V$ represents the set of labeled nodes, $y_v$ is the one-hot encoding of node $v$, and $h_t$ is the vector representation of node $v$. The Pseudo code of DroidNTA Model shows in Algorithm 1.

### *3.5 Interpretability Analysis*

In order to study the feature importance of the model, we analyze the influence of dynamic API features and static network traffic features on the model results by adjusting the influence of parameters ($\alpha$, $\beta$). In addition, to further obtain the importance of each feature to the DroidNTA model in dynamic and static states, we combined the Shapley value to conduct a global interpretability study on the data features. The Shapley value introduced by KernelSHAP in prior work [40], which is a metric used to determine how much each player contributes to the final outcome in a cooperative game.

$$\phi_i(f, x') = \sum_{z' \subseteq \{x'_1,\ldots,x'_n\}\setminus\{x'_i\}} \frac{|z'|!(M-|z'|-1)!}{M!} * [f(z' \cup x'_i) - f(z')] \tag{10}$$

where $x'$ is a subset of the features, $x'_i$ is the vector of feature values of the instance to be explained. $M$ represents the number of features. $f(z')$ is the prediction for feature values in set $z$.

**Algorithm 1:** Pseudo code for DroidNTA model

**Input:** $API\_Graph$: Extract the API to build api graph; $NT\_Data$: Dynamic Network traffic feature;
**Output:** Malware classification $2\_class$; $5\_class$; $43\_class$.

**for** *Feature* in $NT\_Data$ **do**
  $Feature\_Normalization \leftarrow \frac{Feature - Feature_{min}}{Feature_{max} - Feature_{min}}$
  $NT\_Vec.append(Feature\_Normalization)$
**end for**
**for** *API* in $API\_Graph$ **do**
  $Node \leftarrow Abstraction(API)$
  *Nodes.append(Node)*
**end for**
**for** *i* in *Nodes* **do**
  $API\_Graph \leftarrow Node_i + Node_{i-1}$
  $API\_Path \leftarrow Extract(API\_Graph)$
  $API\_Vec \leftarrow Markov(API\_Path)$
**end for**
**function** DroidNTA($NT\_Vec, API\_Vec$)
  $All\_Feature \leftarrow \alpha^{*}NT\_Vec + \beta^{*}API\_Vec$
  $(MalData : 2/5/43\_class) \leftarrow$ Eqs. (7) and (8)
  $Loss(\hat{y}) \leftarrow$ Eq. (9)
  $result \leftarrow torch.max(\hat{y})$
  **return** *result*
**end function**

## 4 Experiment Set up

### 4.1 Dataset

In order to verify the effectiveness of DroidNTA model, we select the publicly available CICAndMal2017 [41] dataset. The CICAndMal2017 is the real world data published by the Canadian Institute for Cybersecurity in 2018. Table 2 shows the overall statistics of the CICAndMal2017 dataset. The dataset collected more than 10854 Android samples, including 4354 malicious software and 6500 benign software. In addition, the researchers installed these samples and used CICFlowMeter (https://www.unb.ca/cic/research/applications.html#CICFlowMeter) tool to monitor these Android applications in real time. The 5-class primarily comprise Adware, Ransomware, SMS Malware, Scareware, and Trojans. The 43-class classification represents a granular refinement of the five primary malware categories. For Example, the Adware category is subdivided into 10 distinct families, including Youmi and Dowgin; In total, this taxonomy encompasses 43 discrete classes. We strictly ensured that the training and testing sets are mutually exclusive; thus, no single application instance appears in both splits.

**Table 2:** CICAndMal2017 dataset statistics.

| Category | Benign | Malware |
|---|---|---|
| APP numbers | 4354 | 6500 |
| APP with traffic data | 1700 | 426 |
| Traffic data | 18.5 G | 19.0 G |
| Permission and intents | 8115 | |

To mitigate the challenges posed by the highly skewed class distribution in the original dataset, we optimized the model's data-loading mechanism by integrating a WeightedRandomSampler strategy. By assigning sampling weights inversely proportional to class frequencies, the framework ensures that minority category features are adequately represented within each training batch. This approach systematically rectifies the inherent prediction bias caused by sample-size disparities in deep learning.

### 4.2 Evaluation Metric

Since the accuracy metric cannot fully reflect the detection performance of the model in the multi classification research problem, we comprehensively considers four metrics. We selected four automation metrics: Accuracy, Precision, Recall and F1 value to verify the validity of the DroidNTA model. The calculation equations of these metrics are as follows:

$$Accuracy = \frac{TP + TN}{FP + FN + TP + TN} \tag{11}$$

$$Precision = \frac{TP}{FP + TP} \tag{12}$$

$$Recall = \frac{TP}{FN + TP} \tag{13}$$

$$F1 = 2 * \frac{Precision \times Recall}{Precision + Recall} \tag{14}$$

The TP/FP/TN/FN respectively denote correctly classified positive, falsely positive, correctly classified negative, and falsely negative samples.

### *4.3 Baselines*

We have selected six models from the previous work for comparison. These baselines can be divided into two categories: 1. Classic machine learning model. 2. Deep learning model.

**KNN** [42] and **SVM** [43]: The two models are traditional machine learning models, which can divide data through supervised learning methods. KNN is a classification method based on the feature space of $K$ similar samples, and the label of test samples is determined by the number of categories in the feature space. The decision boundary of SVM is the maximum margin hyperplane for learning samples.

**DT** [44]: Researchers have proposed a dynamic detection technology with network traffic feature based on decision tree, which records the behavior of Android applications at runtime. The experiment shows that when J48 decision tree algorithm is used for malware detection, the model has better prediction accuracy in Drebin dataset.

**MaMaDroid** [38]: The MaMaDroid is a system based on static feature analysis method. The model abstracts the API call graph executed by Android applications into their classes, packages or classes node, and builds the API sequences obtained from the application call graph as Markov chains for modeling. The MaMaDroid model was evaluated by malicious application data spanning six years.

**DeepAMD** [45]: The DeepAMD model uses a deep neural network to detect real Android malware. DeepAMD model can identify malware using neural networks on static and dynamic layers. On the static layer, DeepAMD achieve the highest accuracy rate for malware classification, with an accuracy rate of 92.5% for malware categories.

**CACNN** [46]: The CACNN model combines static features with neural networks to detect and experimentally tests its effectiveness. The results of the first layer are fed into the second layer, where the CNN and AutoEncoder are connected for applications that detect malware through network traffic features.

**FAGnet** [47]: FAGnet ingeniously leverages Graph Neural Networks to capture the structural features of Android application function call graphs; by employing clustering analysis, it effectively enhances both the detection accuracy and generalization capability for malware.

### *4.4 Experimental Detail*

All experiments are performed on a system with the following specifications: Intel (R) Core (TM) i5-11500 CPU @2.70GHz, 16 GB RAM, and the operating system is Windows 10. The important third-party libraries is Python 3.7 and Pytorch 1.4. The relevant parameters of this experiment are: Batch-size is 5, the number of epochs is 100, the learning rate is 0.0001, the gradient optimizer is Adam, and the ratio of training and testing sets is 80%:20%. The above parameters are basically consistent with the previous deep learning works.

## 5 Evaluation and Discussion

### *5.1 Research Questions*

In order to verify the overall prediction accuracy, multi classification, stability, parameter and feature importance of the model, the following experimental questions are proposed in this section.

**RQ1: What is the effectiveness of DroidNTA model on overall experimental result?**

**RQ2: How stable and time-consuming is the DroidNTA model?**

**RQ4: How do the fusion parameters ($\alpha$, $\beta$) and neural network architecture affect the preformance DroidNTA model?**

**RQ4: Which of these data features has the greatest impact on the experimental results?**

**RQ5: How robust is the DroidNTA model across different datasets?**

### *5.2 Overall Performance*

For RQ1, we compare the overall performance of DroidNTA model in Android malware classification. It can be seen from the results in Table 3 that the overall experimental results of the DroidNTA model are better than the two types of baselines models. Compared with the traditional machine learning model, the experimental results of DroidNTA model are improved significantly. In addition, the model performs better than the deep learning model.

**Table 3:** The overall preformence of the DroidNTA model with the baselines.

| Model | Metric | | | |
|---|---|---|---|---|
| | **Accuracy** | **Recall** | **Precision** | **F1** |
| KNN | 0.8946 | 0.7582 | 0.7840 | 0.7709 |
| SVM | 0.8843 | 0.6373 | 0.8285 | 0.7204 |
| DT | 0.8997 | 0.7407 | 0.7692 | 0.7547 |
| MaMaDroid | 0.9768 | 0.9382 | 0.9500 | 0.9441 |
| DeepAMD | 0.9340 | 0.9320 | 0.9340 | 0.9350 |
| CACNN | 0.9920 | 0.9820 | 0.9530 | 0.9672 |
| FAGnet | 0.9943 | 0.9857 | 0.9761 | 0.9879 |
| DroidNTA (ours) | 0.9974 | 0.9983 | 0.9946 | 0.9964 |

First of all, compared with traditional machine learning models, the results of DroidNTA model in binary classification are significantly improved. Among them, compared with the SVM model, our model has improved the most, DroidNTA model Recall value has increased by more than 36%, and the F1 value has increased by 27%. Compared with the MaMaDroid model which is the best performing machine learning model, the detection performance of the DroidNTA model is also better. In the Accuracy, Recall, Precision and F1 indicators, the DroidNTA model is 2.6%, 6.0%, 4.5% and 5.2% higher than the MaMaDriod model, respectively. Overall, our proposed DroidNTA model has better predictive performance compared to other machine learning and deep learning models in binary malware detection tasks.

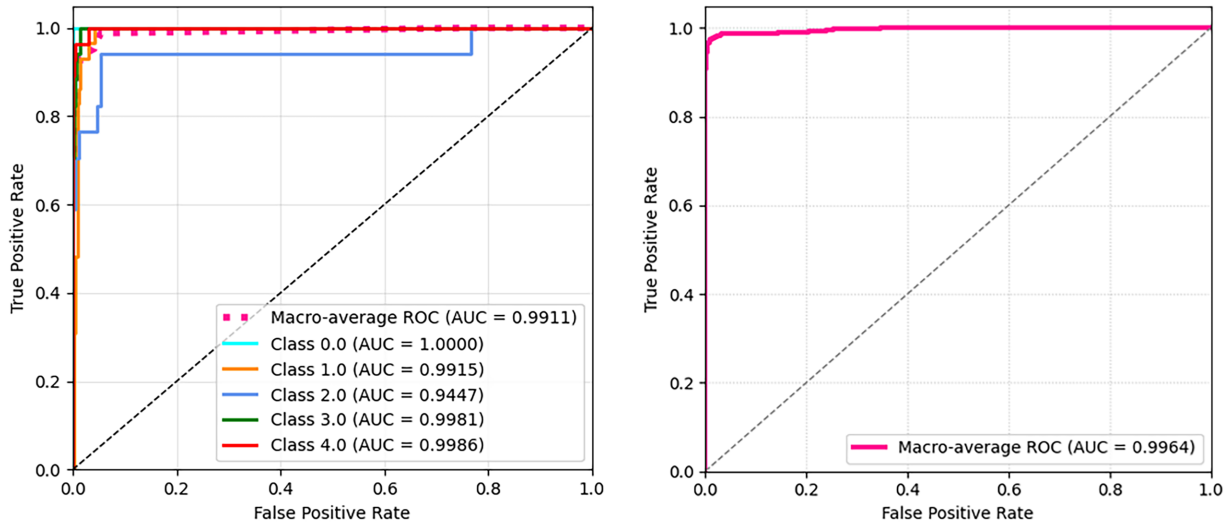
In addition, Combining the categories and family types of malware, the CICAndMal2017 dataset can be subdivided into 5 categories and 43 families malware. To gain insight into the predictive performance of the DroidNTA model, we make predictions on these data and compare with the baseline model.

From the experimental results in Table 4, it can be known that the prediction accuracy rate of the DroidNTA model has 95.89% in the classification results of 5 categories, and the prediction accuracy rate of the model in 43 families is 92.54%. In the 43-class classification results, DroidNTA model prediction performance improvement is obvious, and it is the only detection model with a prediction accuracy of more than 90%. Compared with the best-performing CACNN among deep learning models, our model improves by 21.68%, which is a very large improvement. The DroidNTA model is 1.54% higher than the CACNN model in the 5-class prediction task, it is higher than other models more obvious.

**Table 4:** Multi category performance.

| Model | Classification Scenarios | | |
|---|---|---|---|
| | 2-Class | 5-Class | 43-Class |
| KNN | 0.8946 | 0.8376 | 0.8252 |
| SVM | 0.8843 | 0.8072 | 0.7917 |
| DT | 0.8997 | 0.8771 | 0.8640 |
| MaMaDroid | 0.9768 | 0.8917 | 0.8714 |
| DeepAMD | 0.9340 | 0.8220 | 0.6510 |
| CACNN | 0.9920 | 0.9460 | 0.7215 |
| GNN | 0.9943 | 0.9552 | 0.8726 |
| DroidNTA (ours) | 0.9974 | 0.9614 | 0.9254 |

We plot the ROC curves and calculates the corresponding AUC values for both the 5-class and 43-class classification tasks (as shown in Fig. 3). The experimental results demonstrate that, in the 5-class classification task, the macro-averaged ROC curve achieved an AUC of 0.9911, with the AUC for each individual subclass ranging between 0.9447 and 1.0000; this indicates that the model possesses exceptional discriminative power regarding major malware families. Moreover, in the more challenging 43-class fine-grained family classification task, the AUC further improved to 0.9964, with the curve exhibiting excellent steepness and rapidly converging toward the top-left corner of the graph. This conclusively validates that, when handling large-scale multi-class classification tasks, the DroidNTA model not only delivers superior classification accuracy but also maintains an exceptional balance between sensitivity and specificity, thereby enabling it to effectively address complex and dynamic scenarios involving the discrimination of malware samples.




(a) 5-Class ROC curves and AUC values

(b) 43-Class ROC curves and AUC values

**Figure 3:** ROC curves and AUC values.

Furthermore, to validate the statistical significance of DroidNTA's performance improvements relative to the baseline model, this study conducted both a Paired $t$-test and a Wilcoxon signed-rank test. In the 5-class classification experiment, the $p$-values for both tests reached $1.0 \times 10^{-6}$—far below the 0.01 significance

level demonstrating extremely strong statistical consistency. In the 43-class classification experiment, which featured a more complex sample distribution, the $p$-value for the Paired $t$-test was 0.0000001, while the $p$-value for the Wilcoxon test was 0.001814; both results demonstrated statistically significant differences within the 0.01 confidence interval.

### *5.3 Model Stability and Time Consumption*

The experimental results presented in Table 5 demonstrate that the proposed model exhibits exceptional training stability and favorable convergence characteristics across various task scales. As evidenced by the data, whether applied to a 5-class classification task or a more complex 43-class task, both the variance and standard deviation of the model's performance remained at an extremely low magnitude. Statistically, this provides compelling evidence for the reliability and reproducibility of the experimental results. Specifically, as the number of training epochs increased, various fluctuation metrics displayed a distinct downward trend; notably, in the 5-class classification task, the standard deviation steadily declined from 0.0202 to 0.0011. This reflects the model's ability during the process of fusing API and network features to rapidly overcome perturbations induced by random initialization and transition into a robust state of convergence.

**Table 5:** Variance and standard deviation results across multiple epochs for 5-class and 43-class detection.

| Epoch | Variance | | | | Standard Deviation | | | |
|---|---|---|---|---|---|---|---|---|
| | Epoch-20 | Epoch-40 | Epoch-60 | Epoch-100 | Epoch-20 | Epoch-40 | Epoch-60 | Epoch-80 |
| 5-Class | 0.00041 | 0.000032 | 0.000006 | 0.000001 | 0.0202 | 0.0056 | 0.0024 | 0.0011 |
| 43-Class | 0.000044 | 0.000186 | 0.001811 | 0.000001 | 0.0066 | 0.0136 | 0.0425 | 0.0010 |

The DroidNTA model execution time remained stable at approximately 40 s across various classification scales, with each individual round taking about 0.4 s; the overall results are presented in Table 6. This demonstrates that increasing the granularity of classification did not result in a significant rise in computational overhead, thereby highlighting the architecture's efficiency and low-latency characteristics qualities that make it highly promising for industrial deployment.

**Table 6:** Time consumption in different classification scenarios.

| Model | Classification Scenarios | | |
|---|---|---|---|
| | 2-Class | 5-Class | 43-Class |
| All Time | 40.76 s | 39.82 s | 40.56 |
| Epoch Time | 0.4076 s | 0.3982 s | 0.4056 s |

### *5.4 Parameter Discussion*

For RQ3, we conducted the following two-part experiments to verify the influence of parameters on the DroidNTA model. One is to calculate the influence of network traffic and static API feature fusion parameters ($\alpha$, $\beta$) on the results of the model. The other is the effect of the number of internal nodes of the three-layer

neural network on the experiment. First, we experimented with the DroidNTA model by adjusting the size of the parameters, and the experimental results are shown in Table 7. When the parameter $\alpha = 0$ or $\beta = 0$, the prediction effect of the DroidNTA model is poor. In the 5-class prediction task, when the parameters $\alpha = 0.8$ and $\beta = 0.2$, the prediction performance of the DroidNTA model is the best, and the Accuracy, Recall, Precision and F1 all exceed 0.82. In the 43-class prediction task, when the parameters $\alpha = 0.9$ and $\beta = 0.1$, the prediction performance of the DroidNTA model is the best, and the Accuracy metric reaches 0.9254. From the above results, it can be seen that network traffic feature have a greater impact on the prediction performance of the model.

**Table 7:** The effect of parameters on the experimental results.

| $(\alpha, \beta)$ | 5-Class | | | | 43-Class | | | |
|---|---|---|---|---|---|---|---|---|
| | **Accuracy** | **Recall** | **Precision** | **F1** | **Accuracy** | **Recall** | **Precision** | **F1** |
| (0.0, 1.0) | 0.9460 | 0.7919 | 0.8009 | 0.7910 | 0.8406 | 0.3077 | 0.3041 | 0.2861 |
| (0.1, 0.9) | 0.9537 | 0.8077 | 0.8238 | 0.8085 | 0.8817 | 0.4749 | 0.4275 | 0.4227 |
| (0.2, 0.8) | 0.9537 | 0.8174 | 0.8418 | 0.8235 | 0.8997 | 0.5338 | 0.5143 | 0.5000 |
| (0.3, 0.7) | 0.9486 | 0.8014 | 0.7969 | 0.7968 | 0.9049 | 0.5983 | 0.5151 | 0.5216 |
| (0.4, 0.6) | 0.9486 | 0.8229 | 0.8300 | 0.8165 | 0.8972 | 0.5580 | 0.5016 | 0.4929 |
| (0.5, 0.5) | 0.9486 | 0.8194 | 0.8244 | 0.8164 | 0.9043 | 0.5898 | 0.5179 | 0.5166 |
| (0.6, 0.4) | 0.9486 | 0.8145 | 0.8194 | 0.8134 | 0.9229 | 0.6499 | 0.5877 | 0.5913 |
| (0.7, 0.3) | 0.9537 | 0.8172 | 0.8307 | 0.8213 | 0.9126 | 0.6370 | 0.5627 | 0.5704 |
| (0.8, 0.2) | 0.9614 | 0.8698 | 0.8747 | 0.8697 | 0.9203 | 0.6726 | 0.5985 | 0.6147 |
| (0.9, 0.1) | 0.9383 | 0.7853 | 0.7931 | 0.7800 | 0.9254 | 0.6528 | 0.6315 | 0.6097 |
| (1.0, 0.0) | 0.8406 | 0.5395 | 0.6541 | 0.5766 | 0.8483 | 0.5019 | 0.4703 | 0.4524 |

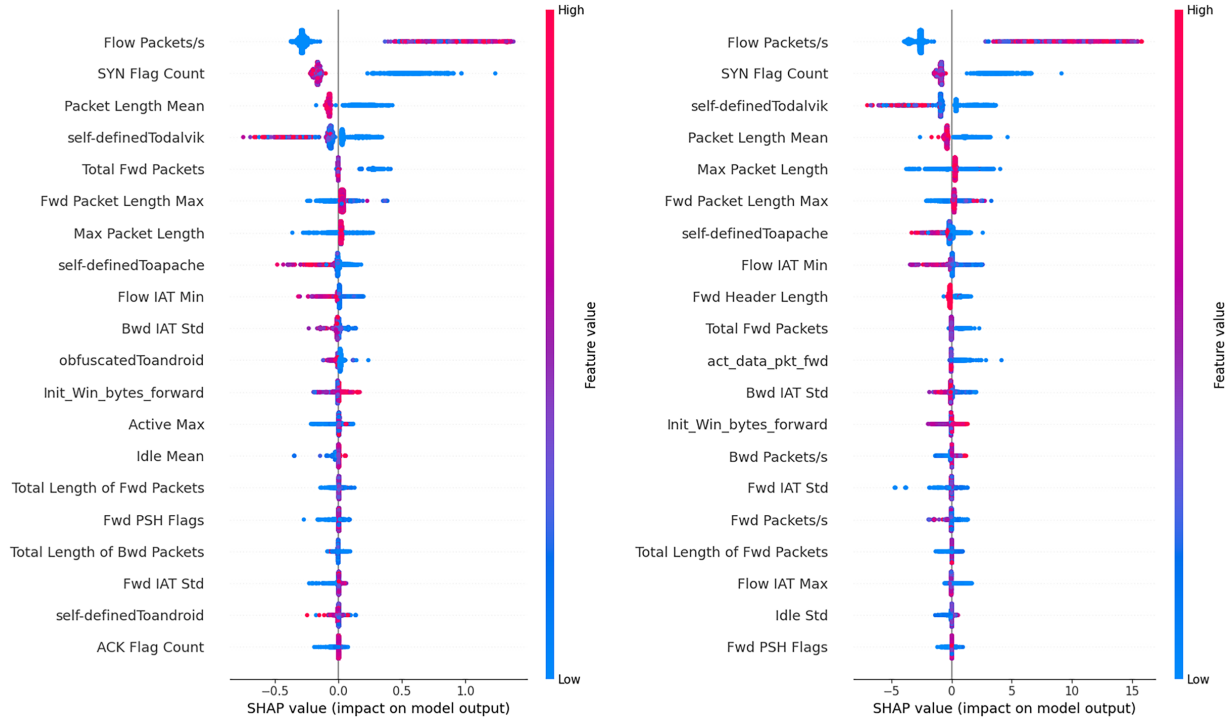
### *5.5 Feature Importance Analysis and Ablation Study*

In order to calculate the impact of different features in the data on the model, we use the SHAP tool to increase the interpretability of the model's global features. By calculating the Shapley value of each feature, the global importance measurement data of the model is obtained, and a SHAP summary map is constructed based on the Shapley value, as shown in Fig. 4.

As can be seen from Fig. 4a, among the top 20 important features in the 5-class prediction task, network traffic features account for 16, while API features only account for 4. Among the top 20 important features in the 43-class prediction task, network traffic features account for 18, while API features only have 2 kinds. The experimental results are similar to the results of the parametric analysis of $\alpha$ and $\beta$. The main reason for this situation is that the feature granularity of the API is relatively coarse, and only a part of the feature structure can be retained. In addition, Flow Packets/s has the greatest impact on the model. The Flow Packets/s feature exerts the most significant impact on the model's predictive output due to its superior efficacy in capturing the automated synchronization and high-cadence behavioral semantics of malware. Obviously, unnecessary network traffic transmission is likely to be malicious software transmitting information in the user mobile phone.

We conducted a rigorous performance profiling of the interpretability module to ensure its feasibility in practical applications. Benchmarking results demonstrate that, utilizing a background dataset comprising 50 samples, DroidNTA requires merely 0.8402 s to compute SHAP values for 10 samples spanning five distinct malware families. During this process, the peak incremental memory usage was a negligible 3.39 MB,

while the overall memory footprint remained consistently stable at approximately 241.55 MB. These metrics conclusively demonstrate that our framework not only delivers high-fidelity behavioral insights but also exhibits near real-time processing capabilities and extremely low resource overhead, rendering it highly suitable for integration into high-throughput automated security auditing systems.




(a) 5-Class Experiment Feature Importance Ranking (b) 43-Class Experiment Feature Importance Ranking

**Figure 4:** Feature importance analysis based on Shapley value.

Table 8 presents the results of the ablation study, indicating that both the feature fusion strategy and the residual network architecture make significant contributions to the model's performance. When either API features or NT features were removed, all model metrics experienced a marked decline, thereby confirming the complementarity of these multimodal features in capturing malicious behaviors. Furthermore, the removal of the residual network resulted in the lowest accuracy score, indicating that a deep architecture is crucial for the nonlinear modeling of complex security features. In summary, the full version of DroidNTA achieved optimal performance across all metrics, validating the necessity of the synergistic interaction among its core components.

**Table 8:** Ablation study of the DroidNTA model on a five-class classification task.

| Model | Classification Scenarios | | |
|---|---|---|---|
| | ACC | Recall | F1 |
| W/O API Feature | 0.9460 | 0.7917 | 0.7910 |
| W/O NT Feature | 0.8483 | 0.5019 | 0.4524 |
| W/O Residual Network | 0.8257 | 0.4732 | 0.4327 |
| DroidNTA Model | 0.9614 | 0.8698 | 0.8697 |

### *5.6 Research on Model Robustness*

For RQ5, to validate the robustness of the DroidNTA model, we incorporated an experimental analysis using the CIC-InvesAndMal2019 dataset. CIC-InvesAndMal2019 is a network traffic dataset released by the Canadian Institute for Cybersecurity, designed for Android malware forensics and detection; it extends and upgrades the data collection framework established by CICAndMal2017. Collected from actual Android devices, this dataset comprises network traffic from 28 distinct malware families as well as benign applications, totaling approximately 23,000 network flow records. The experimental results are presented in Table 9. As indicated by these results, the DroidNTA model by incorporating a more advanced residual network architecture achieves superior detection performance.

**Table 9:** The overall preformence of the DroidNTA model on CIC-InvesAndMal2019 dataset.

| Model | Metric | | | |
|---|---|---|---|---|
| | Accuracy | Recall | Precision | F1 |
| KNN | 0.8506 | 0.7133 | 0.7261 | 0.7358 |
| SVM | 0.8261 | 0.6138 | 0.8057 | 0.7012 |
| DT | 0.8236 | 0.7356 | 0.7217 | 0.7358 |
| MaMaDroid | 0.9567 | 0.9617 | 0.9653 | 0.9682 |
| DeepAMD | 0.9226 | 0.9317 | 0.9273 | 0.9294 |
| CACNN | 0.9635 | 0.9693 | 0.9610 | 0.9687 |
| FAGnet | 0.9609 | 0.9610 | 0.9654 | 0.9522 |
| DroidNTA (ours) | 0.9882 | 0.9868 | 0.9745 | 0.9715 |

## 6 Conclusion and Future Works

In this work, we propose DroidNTA, a multimodal framework fusing static API graphs with dynamic network traffic semantics via Markov chain extraction and SHAP interpretability. Results show that while API structures provide intent context, network traffic specifically packet frequency dominates detection. This suggests runtime communication captures fundamental operational logic more resistant to obfuscation than static code. Future work will focus on: (1) Dataset Scaling, constructing modern, diverse datasets to validate robustness against emerging evasion; and (2) Granular Engineering, exploring sub-routine behaviors to enhance discrimination in complex, encrypted environments.

**Acknowledgement:** None.

**Funding Statement:** This research supported by the Shanghai Sailing Program 24YF2720000.

**Author Contributions:** The authors confirm contribution to the paper as follows: study conception and design: Kang Yang; data collection: Kang Yang; analysis and interpretation of results: Lizhi Cai, Jianhua Wu; draft manuscript preparation: Kang Yang. All authors reviewed and approved the final version of the manuscript.

**Availability of Data and Materials:** The data that support the findings of this study are available from the corresponding author upon reasonable request.

**Ethics Approval:** Not applicable.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References


1. Raja G, Ganapathisubramaniyan A, Anbalagan S, Baskaran SBM, Raja K, Bashir AK. Intelligent reward-based data offloading in next-generation vehicular networks. IEEE Internet Things J. 2020;7(5):3747–58. doi:10.1109/jiot.2020.2974631.
2. Statcounter. Mobile operating system market share worldwide. [cited 2026 Jan 1]. Available from: https://gs.statcounter.com/os-market-share/mobile/worldwide.
3. Rieck K, Holz T, Willems C, Düssel P, Laskov P. Learning and classification of malware behavior. In: Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment, 5th International Conference, DIMVA 2008; 2008 Jul 10–11; Paris, France. p. 108–25.
4. Sami A, Yadegari B, Rahimi H, Peiravian N, Hashemi S, Hamzeh A. Malware detection based on mining API calls. In: Proceedings of the 2010 ACM Symposium on Applied Computing (SAC); 2010 Mar 22–26; Sierre, Switzerland. p. 1020–5.
5. Seraj S, Khodambashi S, Pavlidis M, Polatidis N. HamDroid: permission-based harmful android anti-malware detection using neural networks. Neural Comput Appl. 2022;34(18):15165–74. doi:10.1007/s00521-021-06755-4.
6. Or-Meir O, Nissim N, Elovici Y, Rokach L. Dynamic malware analysis in the modern era-A state of the art survey. ACM Comput Surv. 2019;52(5):1–48.
7. Ucci D, Aniello L, Baldoni R. Survey of machine learning techniques for malware analysis. Comput Secur. 2019;81(4):123–47. doi:10.1016/j.cose.2018.11.001.
8. Athiwaratkun B, Stokes JW. Malware classification with LSTM and GRU language models and a character-level CNN. In: Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017; 2017 Mar 5–9; New Orleans, LA, USA. p. 2482–6.
9. Sudhakar, Kumar S. MCFT-CNN: malware classification with fine-tune convolution neural networks using traditional and transfer learning in Internet of Things. Future Gener Comput Syst. 2021;125(5):334–51. doi:10.1016/j.future.2021.06.029.
10. Moser A, Kruegel C, Kirda E. Limits of static analysis for malware detection. In: Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC 2007); 2007 Dec 10–14, Miami Beach, FL, USA. p. 421–30.
11. Williamson A, Beauparlant M. Malware reverse engineering with large language model for superior code comprehensibility and IoC recommendations. Res Sq. 2024. doi:10.21203/rs.3.rs-4471373/v1.
12. Mani G, Kim M, Bhargava B, Angin P, Deniz A, Pasumarti V. Malware speaks! deep learning based assembly code processing for detecting evasive cryptojacking. IEEE Trans Depend Secur Comput. 2023;21(4):2461–77. doi:10.1109/tdsc.2023.3307445.
13. Halder S, Bewong M, Mahboubi A, Jiang Y, Islam MR, Islam MZ, et al. Malicious package detection using metadata information. In: Proceedings of the ACM Web Conference 2024. New York, NY, USA: ACM; 2024. p. 1779–89.
14. Onieva JA, Jiménez PP, López J. Malware similarity and a new fuzzy hash: Compound Code Block Hash (CCBHash). Comput Secur. 2024;142(1):103856. doi:10.1016/j.cose.2024.103856.
15. Ko SM, Yang J, Kim T, You I. N-gram opcode frequency based malware detection using CNN algorithm. Soft Comput. 2025;29(8):4045–53. doi:10.1007/s00500-025-10659-z.
16. Sharma YK, Tomar DS, Pateriya R, Bhandari S. MOSDroid: obfuscation-resilient android malware detection using multisets of encoded opcode sequences. Comput Secur. 2025;152(4):104379. doi:10.1016/j.cose.2025.104379.
17. Shafiq MZ, Khayam SA, Farooq M. Embedded malware detection using markov n-grams. In: Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment, 5th International Conference, DIMVA 2008; 2008 Jul 10–11; Paris, France. p. 88–107.
18. Ni S, Qian Q, Zhang R. Malware identification using visualization images and deep learning. Comput Secur. 2018;77(4):871–85. doi:10.1016/j.cose.2018.04.005.
19. Mi J, Li Q, Han Z, Liao W, Fu J. Graph learning on instruction stream-augmented CFG for malware variant detection. IEEE Trans Inf Forensics Secur. 2025;20:3015–30.
20. Wu P, Gao M, Sun F, Wang X, Pan L. Multi-perspective API call sequence behavior analysis and fusion for malware classification. Comput Secur. 2025;148(8):104177. doi:10.1016/j.cose.2024.104177.

21. Gebrehans G, Ilyas N, Eledlebi K, Lunardi WT, Andreoni M, Yeun CY, et al. Generative adversarial networks for dynamic malware behavior: a comprehensive review, categorization, and analysis. IEEE Trans Artif Intell. 2025;6(8):1955–76.
22. Vouvoutsis V, Casino F, Patsakis C. Beyond the sandbox: leveraging symbolic execution for evasive malware classification. Comput Secur. 2025;149:104193.
23. Chen Z, Zhou L, Liu Q, Meng W, Weng J. Dynamic malware detection based on enhanced semantic API sequence features. Expert Syst Appl. 2026;315(1):131781. doi:10.1016/j.eswa.2026.131781.
24. Tang M, Qian Q. Dynamic API call sequence visualisation for malware classification. IET Inf Secur. 2019;13(4):367–77. doi:10.1049/iet-ifs.2018.5268.
25. Xu X, Zhang X, Zhang Q, Wang Y, Adebisi B, Ohtsuki T, et al. Advancing malware detection in network traffic with self-paced class incremental learning. IEEE Internet Things J. 2024;11(12):21816–26. doi:10.1109/jiot.2024.3376635.
26. Wang S, Chen Z, Zhang L, Yan Q, Yang B, Peng L, et al. TrafficAV: an effective and explainable detection of mobile malware behavior using network traffic. In: Proceedings of the 24th IEEE/ACM International Symposium on Quality of Service, IWQoS 2016; 2016 Jun 20–21; Beijing, China. p. 1–6.
27. Feng J, Shen L, Chen Z, Lei Y, Li H. HGDetector: a hybrid Android malware detection method using network traffic and function call graph. Alex Eng J. 2025;114:30–45.
28. Prasse P, Machlica L, Pevnỳ T, Havelka J, Scheffer T. Malware detection by analysing network traffic with neural networks. In: Proceedings of the 2017 IEEE Security and Privacy Workshops (SPW); 2017 May 25; San Jose, CA, USA. p. 205–10.
29. Gu J, Zhu H, Han Z, Li X, Zhao J. GSEDroid: GNN-based Android malware detection framework using lightweight semantic embedding. Comput Secur. 2024;140:103807. doi:10.2139/ssrn.4656881.
30. Kunwar P, Aryal K, Gupta M, Abdelsalam M, Bertino E. Sok: leveraging transformers for malware analysis. IEEE Trans Dependable Secur Comput. 2025;22(6):5888–905. doi:10.1109/tdsc.2025.3576708.
31. Dong S, Shu L, Nie S. Android malware detection method based on CNN and DNN bybrid mechanism. IEEE Trans Dependable Secur Comput. 2024;20(5):7744–53. doi:10.1109/tii.2024.3363016.
32. Shu L, Dong S, Su H, Huang J. Android malware detection methods based on convolutional neural network: a survey. IEEE Trans Emerg Top Comput Intell. 2023;7(5):1330–50. doi:10.1109/tetci.2023.3281833.
33. Lu X, Zhao J, Zhu S, Lio P. SNDGCN: robust Android malware detection based on subgraph network and denoising GCN network. Expert Syst Appl. 2024;250:123922.
34. Lin HC, Wang P, Lin WH, Lin YH, Yu YS, Dai JH. Using graph neural network to ransomware detection for cyber threats. In: Proceedings of the 2024 10th International Conference on Applied System Innovation (ICASI); 2024 Apr 17–21. p. 314–6.
35. Wang F, Shi X, Yang F, Song R, Li Q, Tan Z, et al. Malsort: lightweight and efficient image-based malware classification using masked self-supervised framework with swin transformer. J Inf Secur Appl. 2024;83:103784.
36. Yao D, Shao Y. A data efficient transformer based on Swin transformer. Vis Comput. 2024;40(4):2589–98. doi:10.1007/s00371-023-02939-2.
37. Zhou H, Zhang W, Wei F, Chen Y. Analysis of Android malware family characteristic based on isomorphism of sensitive API call graph. In: Proceedings of the Second IEEE International Conference on Data Science in Cyberspace, DSC 2017; 2017 Jun 26–29; Shenzhen, China. p. 319–27.
38. Onwuzurike L, Mariconti E, Andriotis P, Cristofaro ED, Ross GJ, Stringhini G. MaMaDroid: detecting Android malware by building markov chains of behavioral models (extended version). ACM Trans Priv Secur. 2019;22(2):1–34.
39. Norris JR, Norris JR. Markov chains. Cambridge, UK: Cambridge University Press; 1998.
40. Aydoğan B, Aytekin T. An in-depth analysis of KernelSHAP and SamplingSHAP: assessing robustness, error, and efficiency. Knowl Inf Syst. 2025;67(11):10545–79. doi:10.1007/s10115-025-02541-z.
41. Lashkari AH, Kadir AFA, Taheri L, Ghorbani AA. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In: Proceedings of the 2018 International Carnahan Conference on Security Technology, ICCST 2018; 2018 Oct 22–25; Montreal, QC, Canada. p. 1–7.

42. Guo G, Wang H, Bell DA, Bi Y, Greer K. KNN model-based approach in classification. In: Proceedings of the on the Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE—OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003; 2003 Nov 3–7; Catania, Sicily, Italy. p. 986–96.
43. Schüldt C, Laptev I, Caputo B. Recognizing human actions: a local SVM approach. In: Proceedings of the 17th International Conference on Pattern Recognition, ICPR 2004; 2004 Aug 23–26; Cambridge, UK. p. 32–6.
44. Zulkifli A, Hamid IRA, Shah WM, Abdullah Z. Android malware detection based on network traffic using decision tree algorithm. In: Proceedings of the Third International Conference on Soft Computing and Data Mining (SCDM 2018); 2018 Feb 6–7; Johor, Malaysia. p. 485–94.
45. Imtiaz S, ur Rehman S, Javed A, Jalil Z, Liu X, Alnumay WS. DeepAMD: detection and identification of Android malware using high-efficient deep artificial neural network. Future Gener Comput Syst. 2021;115(5):844–56. doi:10.1016/j.future.2020.10.008.
46. Feng J, Shen L, Chen Z, Wang Y, Li H. A two-layer deep learning method for android malware detection using network traffic. IEEE Access. 2020;8:125786–96. doi:10.1109/access.2020.3008081.
47. Wang Z, Zeng K, Wang J, Li D. FAGnet: family-aware-based android malware analysis using graph neural network. Knowl-Based Syst. 2024;289:111531.