Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.051916

ARTICLE

A GAN-EfficientNet-Based Traceability Method for Malicious Code Variant Families

Li Li*, Qing Zhang and Youran Kong

School of Computer and Control Engineering, Northeast Forestry University, Harbin, 150040, China

*Corresponding Author: Li Li. Email: lli@nefu.edu.cn

Received: 18 March 2024 Accepted: 17 May 2024 Published: 18 July 2024

ABSTRACT

Due to the diversity and unpredictability of changes in malicious code, studying the traceability of variant families remains challenging. In this paper, we propose a GAN-EfficientNetV2-based method for tracing families of malicious code variants. This method leverages the similarity in layouts and textures between images of malicious code variants from the same source and their original family of malicious code images. The method includes a lightweight classifier and a simulator. The classifier utilizes the enhanced EfficientNetV2 to categorize malicious code images and can be easily deployed on mobile, embedded, and other devices. The simulator utilizes an enhanced generative adversarial network to simulate different variants of malicious code and generates datasets to validate the model's performance. This process helps identify model vulnerabilities and security risks, facilitating model enhancement and development. The classifier achieves 98.61% and 97.59% accuracy on the MMCC dataset and Malevis dataset, respectively. The simulator's generated image of malicious code variants has an FID value of 155.44 and an IS value of 1.72 ± 0.42. The classifier's accuracy for tracing the family of malicious code variants is as high as 90.29%, surpassing that of mainstream neural network models. This meets the current demand for high generalization and anti-obfuscation abilities in malicious code classification models due to the rapid evolution of malicious code.

KEYWORDS

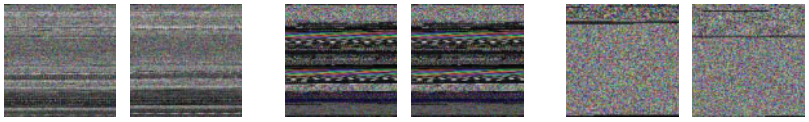
Malicious code variant traceability; feature reuse; lightweight neural networks; code visualization; attention mechanism

1 Introduction

The Security Bulletin: Annual Statistical Report, released by Kaspersky Lab on 14 December, 2023, shows that cybercriminals released an average of 411,000 malicious files per day in 2023, which is a 3% increase from the previous year. During the COVID-19 pandemic, people started using the Internet for communication, online learning, telecommuting, and the expansion of network areas, making it more vulnerable to cybercriminals. These individuals employ new phishing attacks, malware attacks, and other tactics to steal personal and company information, posing a significant threat to the security of individuals' assets. Therefore, addressing the issue of cyberattacks cannot be delayed.

The research on early detection and analysis of malicious code necessitated the artificial extraction of malicious code features. This process required researchers to possess substantial expertise in assembly code and analyze a large number of code snippets, which proved to be costly and inefficient. With the advent of visualization techniques, researchers have transformed malicious code binaries into images. They have utilized feature engineering combined with machine learning techniques to classify them based on their features. This approach reduces labor costs but makes it challenging to identify hidden or obfuscated malicious code variants. With the emergence of deep learning, convolutional neural networks can automatically extract features from malicious code images. These models demonstrate a certain level of generalization capability and anti-obfuscation capability. Despite the gradual development of various types of high-performance deep learning models, accurately combating and timely defending against malicious code variants remains a significant challenge in maintaining global cybersecurity.

Due to the diversity and rapid evolution of malicious code variants, their malicious behavior, structure, and characteristics differ from those of the original version. Consequently, the detection model may not accurately identify and classify them. Some malicious code can mutate itself to evade detection and analysis by altering its own code, potentially causing system damage and service interruptions. However, variants are essentially changes made based on the original version. To prevent and promptly address constantly evolving new types of malicious code, it is essential to study the traceability of malicious code variants. By analyzing malware family relationships, Reference [1] found a significant amount of similar code shared among almost all malware families. Reference [2] utilized image processing techniques to visualize malicious code binaries, as depicted in Fig. 1. Various families of malicious code images from the MMCC dataset and Malevis dataset are showcased. The figure illustrates that malicious code graphs from the same family exhibit similar layouts and textures, while the malicious code images from different families display noticeable differences. Malicious code files are diverse and cannot be quickly compared. The traditional method can only be used for passive analysis, which is much slower than the rate at which malicious code changes. Consequently, it is challenging to grasp and predict the patterns of change in malicious code. Therefore, this paper proposes to begin by focusing on the features of malicious code images, utilizing similar malicious code images with homology, and generating simulated malicious code variants to serve as the data for model training and validation. On the one hand, it predicts the various directions of change in malicious code from a feature perspective. On the other hand, it studies the classification model of malicious code with stronger generalization capability, which can identify various malicious code variants. The main contributions of the work presented in this paper are as follows:



(a) Amonetize (b) Regrun (c) Tracur

Figure 1: Malicious code image samples from different families (a), (b), (c)

1. An effective method for classifying malicious code based on visual features is proposed. This method transforms malicious code into RGB images without the need for feature engineering or disassembly tools. The model automatically extracts malicious code image features and analyzes them for family attributes.

2. A lightweight model is proposed. The Ghost module is utilized to reconstruct the deep convolutional structure in order to reduce the generation of redundant features. This effectively avoids the impact of redundant features on the model size and computational effort.
3. A model with strong generalization and anti-obfuscation capability is proposed. A limited number of essential features are extracted through the ECA-Net structure. Feature reuse is employed to link shallow features with deeper ones, enhancing feature utilization and consequently boosting the model's classification performance.
4. It is proposed to use the improved Auxiliary Classifier Generative Adversarial Network (ACGAN) to simulate images of malicious code variants. This will aid in predicting and simulating changes in the features of malicious code, thereby validating the model's performance and enhancing research on the model.

The structure of the paper is as follows: Section 2 analyzes the methods used for visual feature-based malicious code classification in recent years, Section 3 details the model proposed in this paper, and Section 4 describes the dataset, evaluation metrics, experimental environment, and experimental content for verifying the model's performance.

2 Literature Survey

Malicious code detection and classification have been key research areas. To effectively respond to the threat of malicious code, researchers have recently proposed a vision-based method for detecting malicious code. This method involves transforming a malicious code binary file into an image to distinguish between different families of malicious code based on image features. This section surveys the research on vision-based methods for tracing malicious code variant families using lightweight models. Table 1 presents the methodological characteristics and limitations of the studies in this field.

Table 1: Summary and comparison of existing methods

Author	Date	Method	Characteristic	Disadvantage
Wang et al. [3]	2020/01	VGGNet	Reusability of malicious code	Not validated with an unknown dataset
Li et al. [4]	2021/05	Ghost-DenseNet-SE	Reconstructing DenseNet	The model consumes a significant amount of memory
Umer et al. [5]	2022/01	GANs	Creating morphing samples	Low accuracy in case of attack
Anandhi et al. [6]	2022/08	DenseNet	Generate adversarial attacks	Reduced model accuracy due to interference
Chaganti et al. [7]	2022/09	EfficientNetB1	Multiple representations of malicious code signatures	EfficientNetV1 has drawbacks

(Continued)

Table 1 (continued)

Author	Date	Method	Characteristic	Disadvantage
Gupta et al. [8]	2022/11	ANN	IDA + Feature engineering	The classification accuracy is too low
Wang et al. [9]	2023/01	FFSE	Feature fusion + Channel attention mechanism	Lack of generalization validation
Dao et al. [10]	2023/03	MLP-Mixer-Autoencoder	Multilayer perceptron + Autoencoder	Poor feature learning in models
Liu [11]	2023/08	PV-DM	Open-set classification	Few unknown malicious code samples
Ravi et al. [12]	2023/09	ViT4Mal	Lightweight vision transformer	Model performance degradation

2.1 Traceability Methods for Malicious Code Variant Families

To improve the model's capacity to identify variations of malicious code, researchers have enhanced its generalization capability and anti-obfuscation ability through adversarial attacks and data augmentation. Reference [3] introduced a VGGNet-based classification system for malicious code variants, which can extract more feature information by training with RGB images of malicious code. However, the model was not evaluated with malicious code variants, which does not adequately demonstrate its ability to detect such variants. Reference [5] proposed the use of a GAN to generate zero-day adversarial attacks, providing a method to create attack adversaries using a GAN. However, the detection accuracy of their proposed classification model was only 84%. Reference [6] utilized the fast gradient sign method and additive noise to create adversarial samples for evaluating the detection capability of the DenseNet model. This study validated the robustness of the deep learning model against adversarial attacks. Reference [9] proposed a feature fusion-based fast detection method for malicious code to address the timeliness issue of detecting malicious code variants. However, the test was conducted using the original dataset, which lacked the ability to validate the generalization capability of the model. Reference [11] proposed open-set recognition of malicious code through homology analysis, which can identify unknown categories of malicious code, even with limited samples of such code.

2.2 Lightweight Malicious Code Classification Model

Mobile devices, such as cell phones and laptops, that contain a large amount of personal information and sensitive data can easily become the target of attackers. Therefore, we have developed lightweight malicious code classification models that can be utilized in resource-constrained environments to enhance the information security of a larger number of users. Reference [4] proposed a malicious code detection method based on Ghost-DenseNet-SE, utilizing a lightweight Ghost module to reconfigure the convolutional layer of the Dense block. The enhanced model has a high capacity for feature extraction. However, DenseNet incurs a significant memory overhead during training, and the SE attention mechanism generates a large number of parameters during the modeling process. Reference [7] utilized multiple models to evaluate various malicious code samples. The study found that the lightweight model EfficientNetB1 performed the best on the malicious code dataset.

Moreover, it was concluded that byte-level malicious code images more accurately represent the characteristics of malicious code. Reference [10] proposed an integrated architecture of Multilayer Perceptron and Autoencoder instead of a CNN model. This architecture has fewer hyperparameters and can still operate without a GPU. Reference [12] proposed that the utilization of lightweight vision transformers for malware detection on edge devices, without relying on deeper networks, can enhance computational efficiency. However, the accuracy of lightweight model recognition is not sufficiently high.

3 Methodology

Fig. 2 illustrates the overall framework of the methodology presented in this paper. First, the binary sequence containing the malicious code is divided into 8-bit subsequences. These subsequences are then converted to decimal values ranging from 0 to 255. These decimal values correspond to the R, G, and B pixel points, respectively, to create the RGB map of the malicious code. The images are divided into a training set, a validation set, and a test set in the ratio of 8:1:1. During the training phase, the data is imported into the classifier and the simulator for feature learning of various families. In the testing phase, the test set is fed with a dataset of malicious code variants generated by the simulator into the classifier to evaluate its performance and analyze any weaknesses.

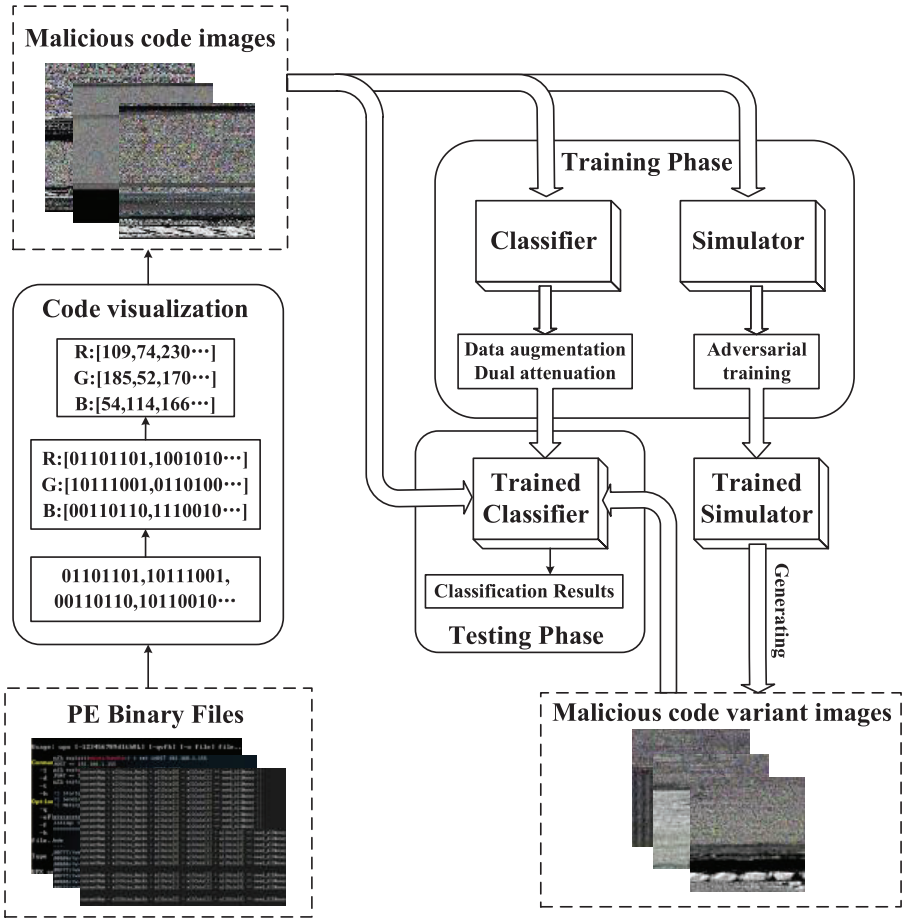


Figure 2: The overall framework of the methodology in this paper

3.1 The Classifier: Improved EfficientNetV2-s

The backbone network of the malicious code variant classifier is the EfficientNetV2-s network. The original model mainly consists of the MB module and Fused-MB module, while the improved model adds a Dense module to connect the shallow features with the deep features. The classifier layer structure is shown in Fig. 3.

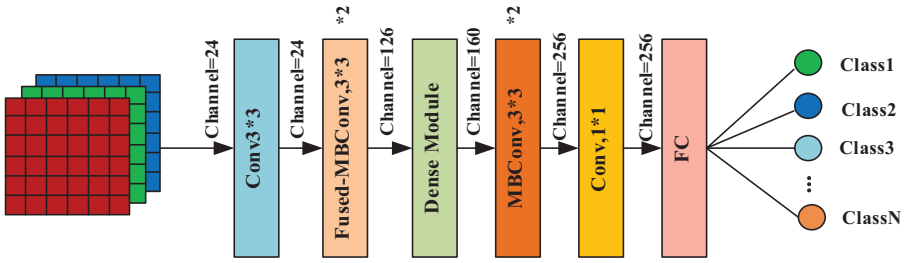


Figure 3: Layer structure of the classifier

3.1.1 Fused MBConv Module

The EfficientNetV2 model combines 1 × 1 pointwise convolution with depthwise convolution to form a standard 3 × 3 convolution. This increases the number of parameters in the model, although it enhances training speed. Therefore, this paper proposes using the Ghost module to reconstruct the convolution layers of the Fused MBConv module, as shown in Fig. 4. The Ghost module utilizes a series of linear transformations to produce feature maps that are equivalent in size to those generated by a standard convolution, but with reduced computational complexity and a smaller parameter count. When expansion $\neq$ 1, two layers of Ghost are used instead of the original 3 × 3 convolution and 1 × 1 convolution layers for dimensionality increase and decrease operations. The SiLU activation function is replaced with the ReLU function, which is more suitable for the nonlinear activation of Ghost, to reduce the transmission and calculation of redundant information. When the expansion factor is set to 1, the original 3 × 3 convolutional structure is preserved and utilized in the shallowest layer of the network. This ensures that the model can achieve quicker inference and computation by working with basic features and patterns.

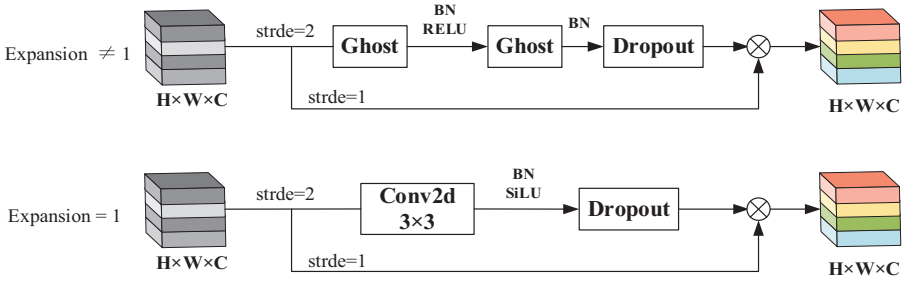


Figure 4: Structure of the improved fused MBConv module

3.1.2 Dense Module

DenseNet utilizes feature concatenation and can achieve excellent results on unknown data with robust generalization capability. We incorporate the Dense module into the middle layer of the

classifier to connect the outputs of the shallow layers to the inputs of all subsequent layers. This ensures effective feature transfer during the training process, reducing computational losses and errors in the model. The Dense module is mainly composed of 1 × 1 convolutions and 3 × 3 convolutions, which are combined using 1 × 1 convolutions and 2 × 2 AvgPooling. The structure is illustrated in Fig. 5. However, having too many densely connected networks can render the model structure redundant and consume a significant amount of memory. Therefore, this paper suggests incorporating only two Dense modules in the middle layer. The first module, DenseBlock, contains three DenseLayers. The second module, DenseBlock, contains six DenseLayers. The Transition layer is used to compress the feature map and connect the two DenseBlocks. The shallow features are passed to the subsequent MBConv module after all the layers are densely interconnected.

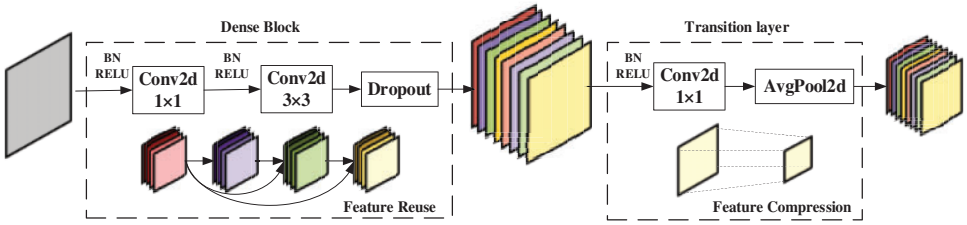


Figure 5: Structure of Dense module

3.1.3 MBConv Module

The MBConv module is situated in the deep layers of the EfficientNetV2 model. It utilizes depth-separable convolution along with Squeeze-and-Excitation Networks (SE) to extract more abstract and advanced features in the deeper layers of the network. To simplify the underlying structure, we suggest utilizing the Ghost module instead of the standard convolution, and the Efficient Channel Attention Network (ECA-Net) instead of the SE. The improved structure is shown in Fig. 6. When stride = 1, the information from the input layer is directly passed to the output layer through a shortcut branch, and 20% of the feature information is randomly discarded. This helps avoid the problem of gradient vanishing caused by the network layers being too deep. When stride = 2, the feature map first passes through a Ghost module, where the output channel number is an inflated multiple of the input channel number. This is followed by depth-separable convolution with the ECA-Net, a process that maintains the same number of channels. Finally, the Ghost module is utilized to scale the feature map to a specified output channel size. Utilizing two Ghost modules for dimensionality enhancement and dimensionality reduction can enhance the model's focus on important features and capture global features, thereby improving the performance, efficiency, and generalization ability of the classifier.

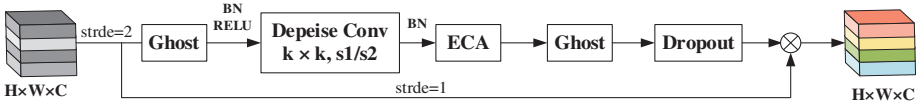


Figure 6: Structure of the improved MBConv module

3.2 Malicious Code Variant Simulator: Improved ACGAN

The backbone network of the malicious code variant simulator is ACGAN, as shown in Fig. 7. The generator includes class labels as auxiliary information for generating images. These labels are fed back to the generator as targets for learning when the discriminator outputs. The discriminator receives the

samples, judges whether they are true or false, and identifies the class labels. This process assists the ACGAN in efficiently simulating data based on different classes simultaneously. To ensure that the ACGAN model realistically simulates malicious code variants, this paper improves the ACGAN model. The simulator structure is shown in Fig. 8.

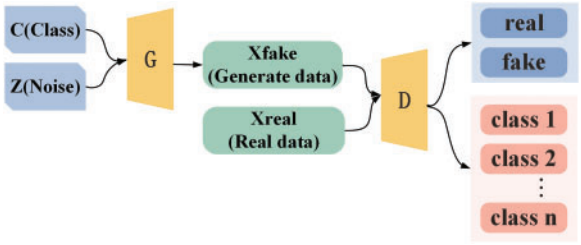


Figure 7: Structure of the ACGAN

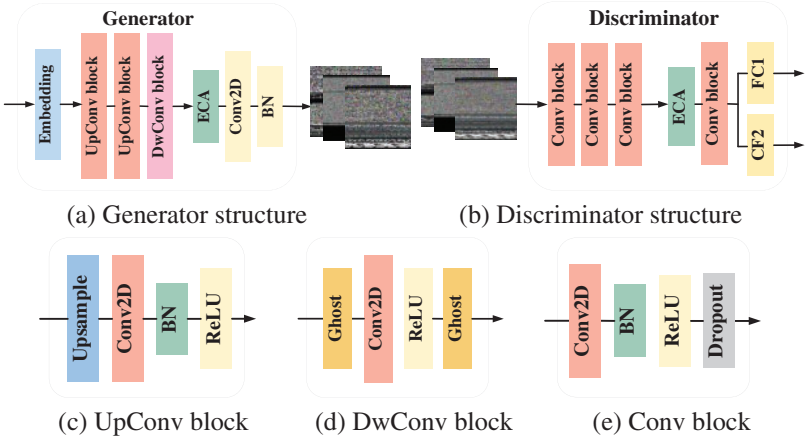


Figure 8: Structure of the simulator

Replacing the generator backbone structure. The original network structure of the generator in the ACGAN is prone to information stacking in the generated image when using deconvolution. This issue significantly impacts the quality of the generated picture. To address this, we have replaced the deconvolution structure into a combination of upsampling and convolution (e.g., Fig. 8c). This modification effectively eliminates the checkerboard pattern in the generated image and enhances the overall quality of the generated images.

Addition of the ECA-Net. This architecture can incorporate a lightweight attention mechanism to each channel to dynamically adjust the feature responses across channels. Due to the absence of an attention mechanism in the original ACGAN, it lacks focus on key regions, leading to the generation of blurry images. Therefore, we propose adding ECA-Net after the embedding layer of the generator and before the output layer of the discriminator, as illustrated in Fig. 8a,b, to assist the model in capturing crucial features and distinguishing features of malicious code images across various categories. This enhancement aims to provide the generator with more precise feedback to simulate images of malicious code variants accurately.

Regulating the capabilities of the generator and discriminator. In ACGAN, it is common for the discriminator to have excessive discriminative power, which can hinder the effective training

of the generator. Therefore, we propose adding a Ghost module for deep convolution reconstruction to the generator (as shown in Fig. 8d) to enhance the generator's learning ability. Additionally, we suggest implementing a random deactivation algorithm after each convolution layer of the discriminator (as illustrated in Fig. 8e) to prevent its ability from growing too rapidly. This approach ensures that both components can compete on an equal footing, facilitating optimization and ultimately reaching a Nash equilibrium.

4 Experiments

4.1 Experimental Environment and Dataset

The development tools and versions used for the experiments in this paper are Python 3.7 and PyTorch 1.10.1. The running server is the Dawning Technology DCU (Name: Heterogeneous Node-4D1-2, Resource No. 7285-32C-128G-4Card-2).

To evaluate the performance of the GAN-EfficientNet model proposed in this paper, we utilized the Microsoft Malware Classification Challenge (MMCC) dataset provided by Microsoft. This dataset comprises nine families of malicious code, totaling 10,868 samples. Moreover, this paper uses the Malevis dataset to verify the domain adaptive capability of the model. The Malevis dataset includes 25 categories of malicious code software and 1 category of cleaning software. In this paper, we selected 25 malware data points to participate in training and testing the model. The number and categories of the two datasets are shown in Fig. 9, and they exhibit different relationships in terms of sample size, categories, and category numbers.

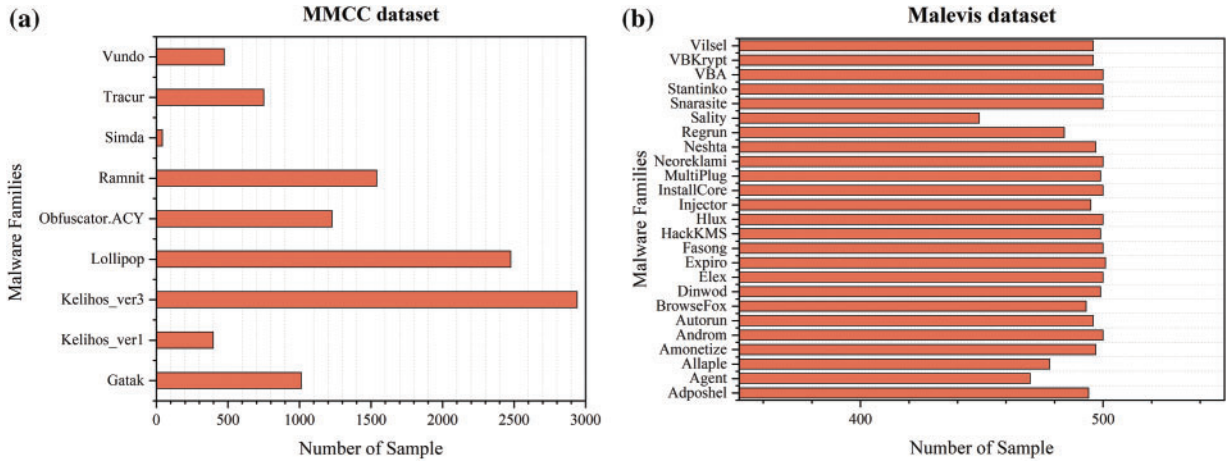


Figure 9: Distribution of malicious code categories and numbers on the (a) MMCC dataset, and (b) Malevis dataset

4.2 Indicators for Experimental Evaluation

4.2.1 Indicators for the Evaluation of the Classifier

In this paper, four metrics—Accuracy, Precision, Recall, and F1-score—are used as indicators of model classification performance. In this case, the metrics use True Positives (TP), False Positives

(FP), True Negatives (TN) and False Negatives (FN) to define the above four metrics, as in Eqs. (1)–(4), respectively.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 - score = 2 \times \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

To evaluate the lightweight and computing power of the model, this paper also considers the number of parameters and the computational load, as shown in Eqs. (5) and (6). The number of parameters indicates the total number of parameters that the model needs to be trained, and the fewer parameters there are, the less memory the model occupies. The computational amount refers to the number of floating-point operations needed to run the network model once. The smaller the computational amount, the faster the computational speed of the model.

$$Parameter = (C_{in} \times (K \times K) + 1) \times C_{out} \tag{5}$$

$$FLOPs = 2 \times H \times W \times (C_{in} \times (K \times K) + 1) \times C_{out} \tag{6}$$

4.2.2 Indicators for the Evaluation of the Simulator

In this paper, we utilize the FID (Fréchet Inception Distance) [13] and IS (Inception Score) [14] to assess the quality of the model in generating malicious code variants from various perspectives. The lower the FID score, the more similar the two images are. The formula is shown in Eq. (7), where x denotes the real image, g denotes the generated image, μ denotes the mean, σ denotes the covariance, and Tr denotes the trace of the matrix.

$$FID\,(x, g) = \left\|\mu_x - \mu_g\right\|_2^2 + Tr\left(\sigma_x + \sigma_g - 2\sqrt{\sigma_x \sigma_g}\right) \tag{7}$$

The higher the IS value, the better the quality and diversity of the image. The formula is shown in Eq. (8), where G denotes the generator, E denotes the expectation, x denotes the generated image, y denotes the classification model prediction result, and D_{KL} denotes the scatter between the distributions.

$$IS\,(G) = \exp(\mathrm{E}_{x \sim p_g} D_{KL}\,(p\,(y|x) \parallel p\,(y))) \tag{8}$$

4.3 Experimental Results

To verify the effectiveness of the proposed method, the following experiments were conducted in this paper.

4.3.1 Performance Comparison of the Classifier with the Original Model EfficientNetV2

The training parameters of the model in this paper are shown in Table 2. The classifier training utilizes the Adam optimizer and implements cosine annealing decay with a learning rate based on the performance of the validation set, following the double decay strategy. The CrossEntropyLoss function is used as the loss function for the classifier, and experiments are conducted on the MMCC

dataset. The accuracy *vs.* loss change curves during the training process are shown in Fig. 10. To verify the domain adaptive ability of the model to different malicious code datasets, this paper utilizes various datasets for training and testing. The accuracy *vs.* loss curves on the Malevis dataset are depicted in Fig. 11.

Table 2: Model training parameters

Parameter	Classifier	Simulator
Epoch	300	500
Batch size	32	32
Learning rate	Generator: 0.0004 Discriminator: 0.0002	0.002
Betas	(0.5, 0.999)	(0.5, 0.999)
Image size	256 ∗ 256	256 ∗ 256
Channel	3	3

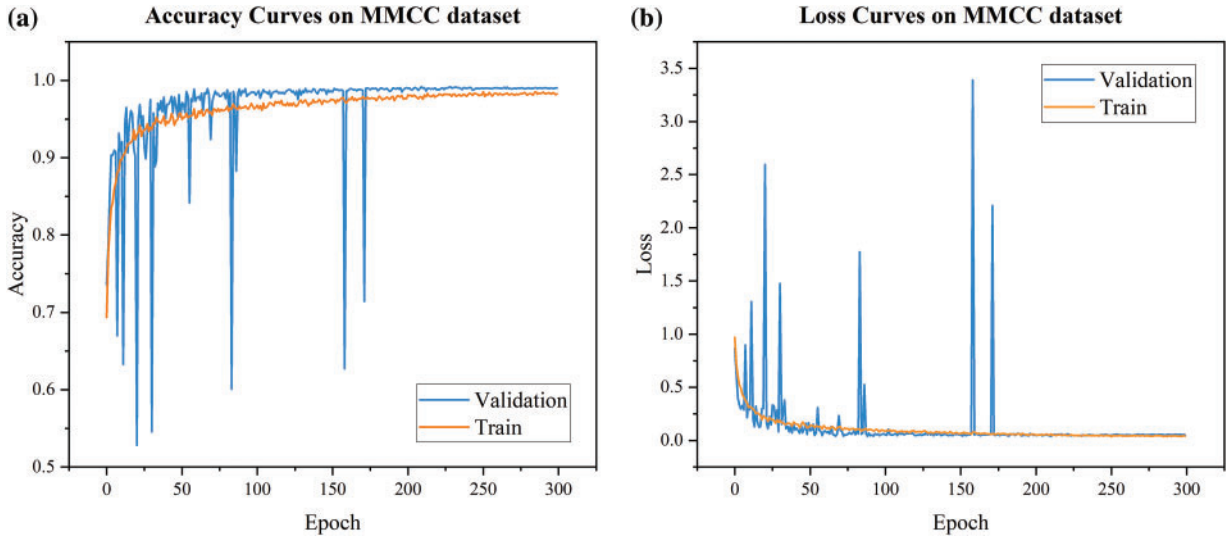


Figure 10: Accuracy and loss curve of the classifier on the MMCC dataset

To verify the effectiveness of this method in improving the model performance and lightening the model, it was compared with the original EfficientNetV2-s model. Table 3 demonstrates four classification indicators, as well as parametric and arithmetic comparisons of the two models on the MMCC dataset and the Malevis dataset.

As shown in Fig. 10, during the initial stage of training on the MMCC dataset, the classifier displayed instability on the validation set. This is due to the high randomness of parameter initialization in the early stage of training. The model did not fully learn the features of the data, resulting in significant fluctuations in accuracy and loss on the validation set. After the 200th epoch, the performance of the model stabilizes on both the training and validation sets. As shown in Table 3, the accuracy, precision, recall, and F1-score of the enhanced model have improved by 0.65%, 0.65%, 0.65%, and 0.67%, respectively. Additionally, only 1 in 12.66 of the parameters of the original model are utilized. As shown in Fig. 11, there are no unstable fluctuations in the classifier performance on the Malvis

dataset, and there is not a significant difference in performance between the training and validation sets. As shown in Table 3, the four classification indicators of the enhanced model have increased by 0.16%, 0.22%, 0.16%, and 0.2%, respectively.

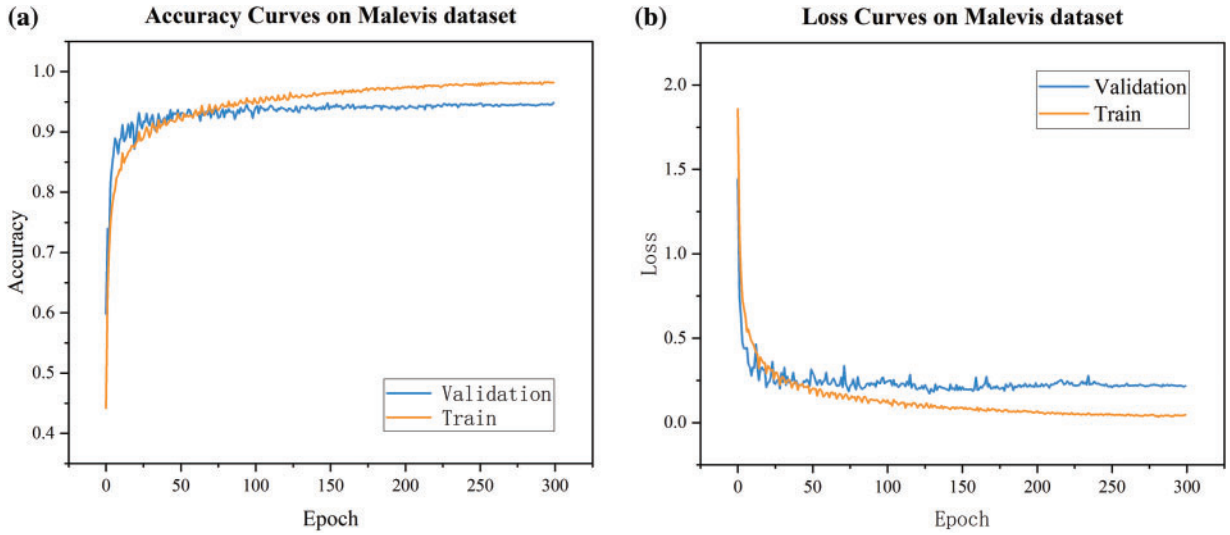


Figure 11: Accuracy and loss curve of the classifier on the Malvis dataset

Table 3: Performance comparison between our method and EfficientNetV2-s

Dataset	Model	Accuracy/%	Precision/%	Recall/%	F1-score/%	Params/M	FLOPs/G
MMCC	EfficientNetV2-s	97.96	98.00	97.96	97.94	20.19	3.75
	Proposed classifier	**98.61**	**98.65**	**98.61**	**98.61**	**1.59**	**4.43**
Malevis	EfficientNetV2-s	97.43	97.49	97.43	97.42	20.19	3.75
	Proposed classifier	**97.59**	**97.71**	**97.59**	**97.62**	**1.59**	**4.43**

The proposed classifier demonstrates good adaptability and generalization ability on both datasets when fewer parameters and fewer operations are utilized. This is because the model is lightweight and utilizes feature reuse methods in the improvement process. Using Ghost instead of regular convolution results in fewer redundant features. Using the ECA-Net structure instead of the SE structure helps avoid introducing a large number of parameters during the modeling process. The Dense module is added to the middle layer for both shallow and deep feature transfer and reuse. Shallow features are fully utilized, reducing feature loss during transmission, and enhancing the model's ability to recognize malicious codes.

4.3.2 Verify the Simulation Quality of Malicious Code Variants

The simulator is trained using the Adam optimizer. The loss function uses Binary Cross Entropy Loss and Focal Loss. The training parameters are shown in Table 2. Fig. 12 displays simulated images of various families of malicious code variants. Column 1 displays the original malicious code images, column 2 shows the images generated by the ACGAN, column 3 presents the images generated by the simulator without utilizing ECA-Net, and column 4 exhibits the images produced by the simulator.

Table 4 corresponds to their FID and IS values, respectively. As shown in Fig. 12 and Table 4, the original ACGAN generated images exhibit a checkerboard pattern, which is due to the stacking of feature information caused by the inability to divide the convolution step size using a deconvolution structure. The simulator, when not utilizing ECA-Net, produces images with inadequate detail characterization and blurriness. Compared to the previous two, the average FID value of the proposed simulator decreased by 218.58 and 33.46, respectively. Additionally, the IS value increased by 0.71 and 0.01, respectively. This indicates a significant enhancement in image quality and diversity. From the experimental results, it can be seen that the generator utilizing a combination of convolution and upsampling structures can prevent the generation of image checkerboards. Additionally, incorporating ECA-Net helps the simulator better capture the details of malicious code images and focus on important features.

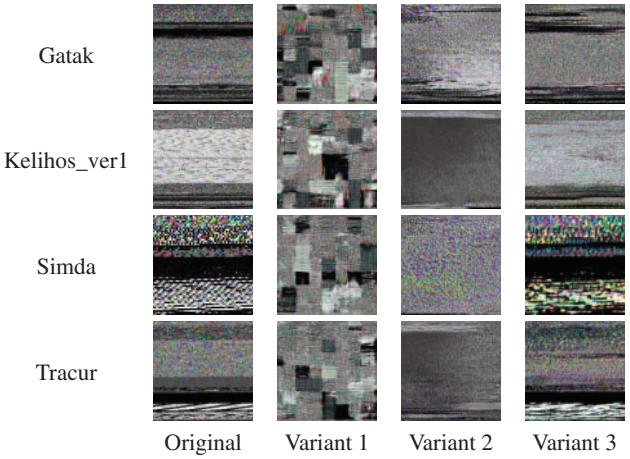


Figure 12: Comparison of generated malicious code variants and original samples

Table 4: Comparison of FID and IS values for samples of different malicious code variants

Dataset	FID	IS
The ACGAN generated dataset	407.48	1.00
The simulator without using ECA-Net generated dataset	188.90	1.71 ± 0.60
Proposed simulator generated dataset	**155.44**	$\mathbf{1.72 \pm 0.42}$
Original dataset	–	1.73 ± 0.54

The variant samples are similar to the original samples but differ in texture, indicating that the simulator can not only retain the features of various malicious code families in the dataset but also makes changes in the detailed portrayal, reflecting the attacker's reuse of the key code in the process of malicious code writing. In summary, the simulator can effectively simulate malicious code variants processed via obfuscation techniques such as rearrangement, encryption and compression.

4.3.3 Performance Comparison between the Proposed Classifier and Other Models

To validate the malicious code classification ability of the classifier proposed, the experiments in this section compare it with seven mainstream neural networks for multiclassification, in addition

to comparing it with models without the addition of the Dense and Ghost structures, respectively, and exploring the role of the two structures. Table 5 shows their performance comparison on the MMCC dataset. Table 5 shows that the accuracy of the proposed classifier in this paper is improved by 1.39%, 1.57%, 0.92%, 0.55%, 5.36%, 5.09% and 1.39% over that of the VGGNet16, GoogLeNet, MobileNetV2, DensnetNet121, EfficientNetB0, EfficientNetB1, and GhostNet models, respectively, by 0.55%, 5.36%, 5.09%, and 1.39%. Moreover, model parameters decreased by 26.11, 4.39, 0.65, 5.37, 2.43, 4.39, and 3.59 million. After the Ghost module has reconstructed the model, the number of parameters and the computation amount are reduced to 93% and 69% of the original, respectively, which can achieve higher classification performance. The Dense module is used to improve the ability of the model to distinguish the features of different categories of malicious codes, and the classification indexes of the model are improved, but this process consumes some computational resources. Reducing the number of parameters facilitates the deployment and operation of the model in restricted environments such as mobile devices or embedded systems. The proposed method increases the computational complexity while improving the model's classification performance and reducing the number of parameters. However, it is far less than the computational complexity of large models. More efficient computational methods or hardware acceleration within an acceptable range can still achieve faster inference speed.

Table 5: Comparison of performance between the model proposed and different models

Type	Model	Accuracy/%	Precision/%	Recall/%	F1-score/%	Params/M	FLOPs/G
Mainstream models	VGGNet16	97.22	97.33	97.22	97.21	27.70	15.42
	GoogLeNet	97.04	97.18	97.04	97.02	5.98	2.08
	MobileNetV2	97.69	97.75	97.69	97.67	2.24	0.42
	DensnetNet121	98.06	98.10	98.06	98.05	6.96	2.88
	EfficientNetB0	93.25	93.73	93.25	93.08	4.02	0.40
	EfficientNetB1	93.52	93.55	93.52	93.28	6.52	0.71
	GhostNet	97.22	97.28	97.22	97.16	5.18	0.20
Models with different structures	The classifier without using Dense	98.33	98.40	98.33	98.33	1.10	1.95
	The classifier without using Ghost	98.43	98.44	98.43	98.42	2.30	4.75
	Proposed classifier	**98.61**	**98.65**	**98.61**	**98.61**	**1.59**	**4.43**

To verify that the proposed classifier can trace the family of malicious code variants, this paper uses the simulator to simulate malicious code variants as a new test set to test the proposed classifier with eight mainstream models and two comparison models. To better characterize the data, the number of individual classes of each family of malicious code variants constructed is the same as that in the original test set. Table 6 shows the different performance values exhibited by the ten models on the new test set. Fig. 13 shows the categorized data of this paper's models on the new test set.

Table 6: Comparison of performance between the model proposed and different models on the new test set

Type	Model	Accuracy/%	Precision/%	Recall/%	F1-score/%
Mainstream models	VGGNet16	80.67	79.73	80.67	79.52
	GoogLeNet	79.28	77.37	79.28	77.24
	MobileNetV2	78.35	78.61	78.35	77.25
	DensnetNet121	76.50	76.60	76.50	78.65
	EfficientNetB0	47.73	47.32	47.73	37.72
	EfficientNetB1	69.10	62.59	69.10	61.44
	EfficientNetV2-s	75.67	75.27	75.67	73.21
	GhostNet	49.31	60.25	49.31	42.50
Models with different structures	The classifier without using Dense	80.31	78.57	80.31	75.57
	The classifier without using Ghost	84.10	82.36	84.10	79.95
	Proposed classifier	**90.29**	**90.42**	**90.29**	**88.62**

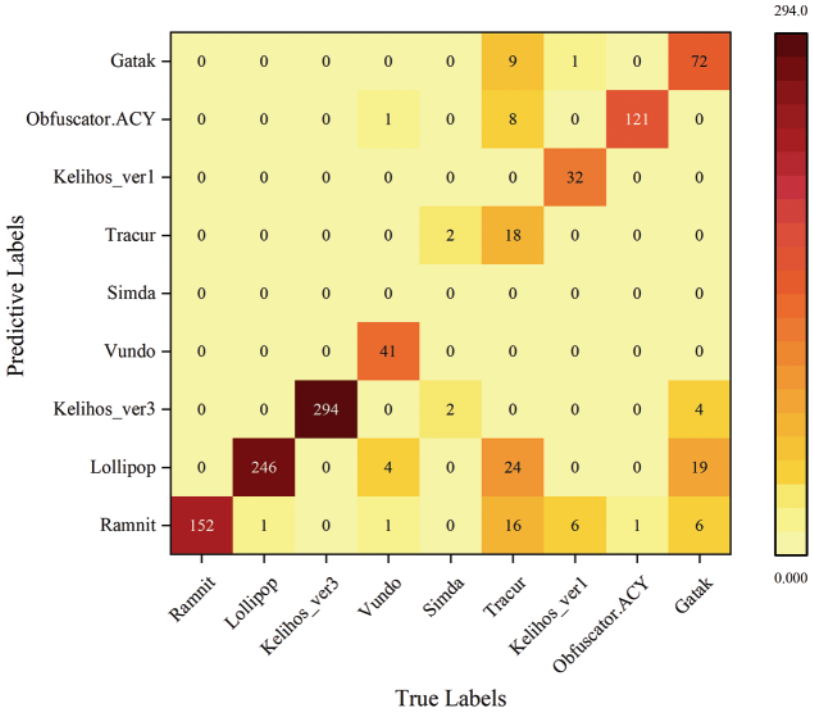


Figure 13: Confusion matrix of the classifier on the new test set

The ten models achieve more than 90% or even greater accuracy on the original test set with strong malicious code categorization capability. However, on the new test set, the accuracies of these models are less than 85%, indicating that these models cannot classify malicious code variants. In contrast, the

classifier proposed in this paper shows excellent performance on both the original and the new test sets. On the new test set, the performance of the proposed classifier is much more excellent than that of the other models. In particular, the classifier we proposed has 9.98% higher accuracy in tracing malicious code variants than the classifier without using the Dense module. This is because the classifier uses the Dense module for feature reuse, which better distinguishes different categories from the feature level. The use of the attention mechanism structure can focus on the critical features of the model to improve the generalization capability of the model. The data enhancement and regularization techniques can improve the model's anti-confusion ability so that even if some of the features of the malicious code have changed to a certain extent, the model can still able to accurately traced back to the malicious code family.

As shown in Fig. 13, all four samples of the Simda family are misclassified, and the classification accuracies of the Tracur family and the Obfuscator.ACY family are also relatively low, indicating that the model is affected by the class imbalance problem of the MMCC dataset, and it is easy to produce incorrect predictions for these types of malicious code family deformations or interferences with a small number of samples. The model can be modelled based on this problem in the future. In the future, the model can be improved according to this problem to solve the impact of the class imbalance problem of the data on the model's performance.

4.3.4 Comparison of the Method Used in This Article with other Studies in the Field

Table 7 compares this paper's method with the research techniques of the last three years, which have achieved significant results. On the MMCC dataset, the accuracy of the proposed method is greater than that of other methods in the literature, and it is an efficient and lightweight model that can ensure that the model can be accurately categorized while requiring fewer parameters and less computation. In image-based malicious code classification, feature extraction is performed by processing image information, and the literature in Table 7 uses grayscale, RGB, and RGBA maps, which are 1-, 3-, and 4-channel images, respectively, input data for the model; these maps contain feature information ranging from less to more. Compared directly, more than the input grayscale map in other studies is needed to satisfy the complete learning of the classification model, and the input RGBA image contains too much information, which reduces the learning efficiency of the classification model. In contrast, the method in this paper focuses on increasing feature information to provide model training while minimizing redundant information to achieve efficient model performance.

Table 7: Comparison of the method used in this article with other studies in the field

Author	Model	Feature	Date	Accuracy/%
Zhu et al. [15]	WGAN-GP	Grayscale image	2021	97.18
Hemalatha et al. [16]	DenseNet	Grayscale image	2021	98.46
Wang et al. [17]	DEAM	Grayscale image	2021	97.30
Shao et al. [18]	Mcs-ResNet	RGBA image	2022	97.21
Anandhi et al. [6]	DenseNet	Grayscale image	2022	96.79
Qiu et al. [19]	HBF-VGG14-Net	Grayscale image	2023	96.00
Liu [11]	PV-DM	Grayscale image	2023	96.00
Proposed	**Classifier**	**RGB image**	**2024**	**98.61**

5 Conclusion

To summarize, this paper proposes a GAN-EfficientNet-based malicious code variant family traceability method consisting of a classifier and simulator. The classifier is a lightweight model with high accuracy, strong generalization capability, and anti-obfuscation ability, and the simulator can learn the critical features of malicious code and simulate malicious code variants. In this paper, the simulator simulates the malicious code variants to effectively verify the tracing ability of the classifier's variant family and determine the shortcomings of the classifier, which points to the direction for the next step to improve the model. The method in this paper will be more targeted according to the characteristic information from different malicious code families, timely strategies for cracking malicious attacks will be developed, and essential means and guarantees for maintaining network security will be provided.

However, there is still room for improvement in the computational efficiency of the classifier proposed. More suitable channel number rules can be determined through experiments and tuning, balancing the model's expressive power with computational resource requirements.

During the research on generative adversarial networks, we found that inputting adversarial samples in model training can improve the sensitivity of the model to feature changes. In the future, we will carry out adversarial training of the model to further improve the classifier's ability to trace the malicious code variant families.

Acknowledgement: Firstly, I sincerely thank my teacher for their careful guidance. Secondly, I would like to thank my classmates in the laboratory for their help. Lastly, I thank academic experts for their valuable feedback and suggestions on my paper.

Funding Statement: The source of funding to support this work is the Key Research and Development Program of Heilongjiang Province, specifically Grant Number 2023ZX02C10.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Qing Zhang, Li Li; data collection: Qing Zhang, Youran Kong; analysis and interpretation of results: Qing Zhang; draft manuscript preparation: Qing Zhang. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: All data used in the experiment can be downloaded from the Kaggle website. BIG 2015 dataset: https://www.kaggle.com/competitions/malware-classification/data (accessed on 15/04/2024). Malevis dataset: https://www.kaggle.com/datasets/sohamkumar1703/malevis-dataset/data (accessed on 15/04/2024).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] D. Zhang and P. Luo, "Review of code similarity detection methods and tools," *Comput. Sci.*, vol. 47, no. 3, pp. 5–10, Aug. 2020. doi: 10.19678/j.issn.1000-3428.0051790.

[2] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Visual Cyber Security*, New York, NY, USA, Jul. 2011, pp. 1–7. doi: 10.1145/2016904.2016908.

[3] B. Wang, H. H. Cai, and Y. Su, "Classification of malicious code variants based on VGGNet," *J. Comput. Appl.*, vol. 40, no. 1, pp. 162–167, Jan. 2020. doi: 10.11772/j.issn.1001-9081.2019050953.
[4] Y. Li and J. Li, "A malicious code detection method based on Ghost-DenseNet-SE," *J. Air Force Eng. Univ.*, vol. 22, no. 5, pp. 49–55, Oct. 2021. doi: 10.3969/j.issn.1009-3516.2021.05.008.
[5] M. Umer, Y. Saleem, M. Saleem, and A. Naqqash, "A GAN based malware adversaries detection model," in *2021 15th Int. Conf. Open Source Syst. Technol. (ICOSST)*, Lahore, Pakistan, Dec. 2021, pp. 1–9. doi: 10.1109/ICOSST53930.2021.9683863.
[6] V. Anandhi, P. Vinod, V. G. Menon, and K. M. Aditya, "Performance evaluation of deep neural network on malware detection: Visual feature approach," *Cluster Comput.*, vol. 25, no. 6, pp. 4601–4615, Aug. 2022. doi: 10.1007/s10586-022-03702-3.
[7] R. Chaganti, V. Ravi, and T. D. Pham, "Image-based malware representation approach with EfficientNet convolutional neural networks for effective malware classification," *J. Inf. Secur. Appl.*, vol. 69, no. 7, pp. 103306, 2022. doi: 10.1016/j.jisa.2022.103306.
[8] K. Gupta, N. Jiwani, M. H. U. Sharif, R. Datta, and N. Afreen, "A neural network approach for Malware classification," in *2022 Int. Conf. Comput., Commun. Intell. Syst. (ICCCIS)*, Greater Noida, India, Nov. 2022, pp. 681–684. doi: 10.1109/ICCCIS56430.2022.10037653.
[9] S. Wang, J. Wang, Y. N. Wang, and Y. F. Song, "A fast detection method for malicious code based on feature fusion," *Acta Electron. Sin.*, vol. 51, no. 1, pp. 57–66, Jan. 2023. doi: 10.12263/DZXB.20211701.
[10] T. V. Dao, H. Sato, and M. Kubo, "MLP-mixer-autoencoder: A lightweight ensemble architecture for malware classification," *Information*, vol. 14, no. 3, pp. 167, Mar. 2023. doi: 10.3390/info14030167.
[11] Y. Q. Liu, "Homology analysis of malicious code families based on open set recognition," *J. Inf. Secur. Res.*, vol. 9, no. 8, pp. 762–770, Aug. 2023. doi: 10.12379/j.issn.2096-1057.2023.08.07.
[12] A. Ravi, V. Chaturvedi, and M. Shafique, "ViT4Mal: Lightweight vision transformer for malware detection on edge devices," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 5, pp. 117:1–117:26, Sep. 2023. doi: 10.1145/3609112.
[13] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," *Adv. Neural Inf. Process. Syst.*, pp. 25–34, Jan. 2019. doi: 10.48550/arXiv.1706.08500.
[14] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, "Improved techniques for training GANs," *Adv. Neural Inf. Process. Syst.*, vol. 29, Jun. 2016. doi: 10.48550/arXiv.1606.03498.
[15] X. H. Zhu, L. P. Qian, and W. Fu, "A method for enhancing malicious code based on generative adversarial networks," *Comput. Eng. Des.*, vol. 11, pp. 3034–3042, Nov. 2021. doi: 10.16208/j.issn1000-7024.2021.11.005.
[16] J. Hemalatha, S. Roseline, S. Geetha, and S. Kadry, "An efficient DenseNet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, pp. 344, Mar. 2021. doi: 10.3390/e23030344.
[17] C. Wang, Z. Zhao, F. W. Wang, and Q. R. Li, "A novel malware detection and family classification scheme for IoT based on DEAM and DenseNet," *Secur. Commun. Netw.*, vol. 2021, pp. 1–16, Jan. 2021. doi: 10.1155/2021/8690662.
[18] Y. Shao, Y. Lu, D. Wei, J. L. Fang, F. W. Qin and B. Chen, "Malicious code classification method based on deep residual network and hybrid attention mechanism for edge security," *Wirel. Commun. Mob. Comput.*, vol. 2022, no. 1, pp. 1–19, Jul. 2022. doi: 10.1155/2022/3301718.
[19] X. L. Qiu, H. M. Zhang, and H. B. Yan, "Quantitative CNN malicious code detection method," *Comput. Integr. Manuf. Syst.*, vol. 40, no. 7, pp. 224–228+295, Jul. 2023.