Journal of Surveillance, Security and Safety

Original Article

Open Access

Graph neural network based function call graph embedding for malware classification

Minami Someya[1,2], Yuhei Otsubo[1,2,3], Akira Otsuka[1]

[1]Institute of Information Security, 2-14-1 Tsuruya-cho, Yokohama, Kanagawa 221-0835, Japan.
[2]National Police Agency, 2-1-2 Kasumigaseki, Chiyoda, Tokyo 100-0013, Japan.
[3]National Police Academy, 3-12-1 Asahi-cho, Fuchu, Tokyo 183-0003, Japan.

Correspondence to: Minami Someya, PhD, Candidate, Institute of Information Security, 2-14-1 Tsuruya-cho, Yokohama, Kanagawa 221-0835, Japan. E-mail: mgs227501@iisec.ac.jp

How to cite this article: Someya M, Otsubo Y, Otsuka A. Graph neural network based function call graph embedding for malware classification. *J Surveill Secur Saf* 2023;4:47-61. http://dx.doi.org/10.20517/jsss.2022.26

Received: 21 Dec 2022 **First Decision:** 8 Jun 2023 **Revised:** 26 Apr 2023 **Accepted:** 26 Jun 2023 **Published:** 30 Jun 2023

Academic Editor: Atsuko Miyaji **Copy Editor:** Yanbin Bai **Production Editor:** Yanbin Bai

Abstract

Aim: Malware family classification is critical for identifying unknown malware species, helping prevent infections, and making analysis more efficient. Existing studies have shown that Function Call Graphs (FCGs) can be used to classify malware types with high performance. However, the effectiveness of FCGs has yet to be thoroughly discussed, and it is unclear how much they contribute to improving the results of malware classification. These questions need to be answered for the future growth of research in this area.

Methods: We propose a malware classification method SAFE+GNN based on the Graph Neural Network (GNN) and FCG. SAFE+GNN updates function features to reflect function call relationships and classify malware. We also developed a comparison model SAFE+MLP without graph structure. We investigate the effects of the FCG structure on malware classification by comparing the experimental results of the two models.

Results: Evaluation experiments using two datasets show that SAFE+GNN was successfully classified with an F1-Score of 98.27% and 98.31%. These results are comparable to world-leading methods and support the effectiveness of using function features in malware classification. Comparing the classification results of SAFE+GNN and SAFE+MLP, we observed that SAFE+GNN tends to classify more accurately than SAFE+MLP for samples with a small number of similar samples in the dataset.

Conclusion: This study confirms the effectiveness of GNNs in malware classification. Future experiments should be conducted on larger datasets that align with real-world scenarios.

Keywords: Malware, family classification, machine learning, graph neural network

1. INTRODUCTION

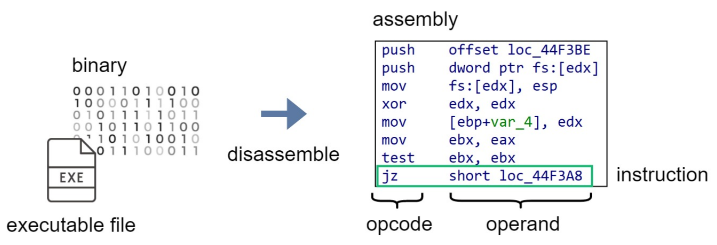
Malware has rapidly increased and has become a significant threat to businesses, organizations, and governments. Malware analysis is an important task to clarify its behavior and case. Static analysis, performed without running malware, is necessary to examine the malware behavior in detail. As shown in Figure 1, the binary executable file is disassembled and converted into assembly instructions in static analysis. The analyst examines the malware behavior by interpreting the instructions consisting of opcodes (operations to be executed) and operands (operation targets). Such analysis requires advanced techniques and much time. Especially when the malware species is unknown, it takes a considerable time.

Against this background, there has been much research into improving the improvement of the efficiency of static malware analysis, and classifying malware families is one of the fields. If we can correctly identify the malware family that represents malware species, it will be easier to infer the malware behavior and improve the analysis efficiency. Attackers create a range of sophisticated malware variants to evade traditional signature-based detectors. Signature-based approaches are limited in responding to variants and new malware in real-time because pattern file creation cannot keep up. Therefore, more advanced malware classification methods are needed to deal with the challenges posed by such new malware types.

Machine learning-based approaches for malware classification have been extensively researched. Machine learning is a way to deal with unknown malware classification. However, machine learning-based methods also have problems. Supervised learning, currently the mainstream method, uses existing samples to learn group patterns. Therefore, a large number of similar samples must be trained in advance. On the other hand, malware samples are updated daily, and sometimes, their characteristics may change even to the same family. After such a major malware update, the learning model needs to be updated by training new samples. In this case, the model needs to correctly classify variants of the new sample even when there are not many learnable new samples.

Graph Neural Network (GNN)[1]-based malware classification methods[2,3] have attracted attention in recent years. A GNN classifies graphs on the basis of graph structures that represent the relationships among objects. To represent a program in a graph structure, a Function Call Graph (FCG) can be used. It is a graph that represents the calling relationship of functions. Recently, GEMAL, an FCG-based malware classifier, was proposed by Wu *et al.*[3]. They showed that GEMAL achieves high performance in classifying malware through their evaluation experiments. However, existing studies have not investigated why GNNs work well and what kinds of samples can be classified using them. Therefore, we use GNNs to classify FCGs and aim to specifically clarify how much FCGs improve the results of malware classification.

Feature extraction is required to represent a function as a vector to learn an FCG with a GNN. Several methods for extracting function semantics have been studied in the field of function similarity comparisons[4,5]. These studies have shown that generating embedding vectors by representation learning noticeably improves performance compared to manually designed features. SAFE by Massarelli *et al.*[4] applied natural language processing techniques to obtain vector representations that capture the semantic information of functions. We expect to represent malware features more accurately using the feature-extracted functions with SAFE.



```
push    offset loc_44F3BE
push    dword ptr fs:[edx]
mov     fs:[edx], esp
xor     edx, edx
mov     [ebp+var_4], edx
mov     ebx, eax
test    ebx, ebx
jz      short loc_44F3A8
```

Figure 1. Disassembly process of an executable file.

In this study, we propose a novel malware classification method SAFE+GNN using GNN and function embedding based on natural language processing techniques. We show that the FCG-based method is effective for malware classification by comparing the results with existing studies. We also investigate the effects of the FCG structure on malware classification by comparing the model SAFE+MLP, which ignores the graph structure.

The contributions of this paper are as follows:

- Our results support the effectiveness of FCGs in malware classification. We evaluated the performance of SAFE+GNN on two datasets and compared them to existing studies. The results show SAFE+GNN successfully classified with F1-Scores of 98.27% and 98.31%, equivalent to world-leading methods.
- We identify a trend of samples in which GNNs excel. We investigate the characteristic of samples that GNNs classify well by comparing the classification results of SAFE+GNN and SAFE+MLP, which does not use the FCG structure. As a result, we obtain the insight that SAFE+GNN tends to classify more accurately compared to SAFE+MLP when there are few similar samples in the dataset.

2. RELATED WORK

Malware analysis involves two primary methods: static and dynamic. Our approach focuses on static analysis, a method that does not execute malware. The objective of our study is to classify malware families. Additionally, we concentrate on malware in the Portable Execution (PE) format, which is a Windows executable file format. Previous work by Ma *et al.*[6] evaluated nine studies on PE malware family classification using four datasets. We use the same datasets to evaluate our experimental results compared to theirs. We will focus mainly on the study used for comparison with our method in the following subsections.

2.1. Binary-based approaches

Binary-based approaches process malware binaries as time series data. The MalConv model proposed by Raff *et al.*[7] is the first end-to-end malware detection model that takes the entire malware executable as input. Qiao *et al.*[8] combined Word2Vec[9] and Multi-Layer Perceptron (MLP) for malware classification. In this method, meaningless bytes are first removed by preprocessing. Then, MLP classifies the 256-byte embedding vector obtained from Word2Vec. Binary-based methods are advantaged by their low preprocessing cost as they use byte sequences. However, they have the problem that jump instructions and function call instructions jump to distant addresses, so adjacent bytes are not necessarily contextually meaningful. This is why we use an FCG to provide a contextually meaningful representation.

2.2. Image-based approaches

Image-based approaches transform malware binaries into color or greyscale images and apply image classification models to classify malware. Nataraj *et al.*[10] originally proposed this method. Ma *et al.*[6] used four representative image classification models (Inception-V3[11], ResNet-50[12], IMCFN[13], and VGG-16[14]) in their comparison experiments. These methods can utilize existing models for image classification and are well-suited to machine learning. However, they have the disadvantage that their features depend on the width of the image and require manual parameter adjustment. Our method does not require manual parameter adjustment for preprocessing because we use learning-based feature extraction.

2.3. Disassembly-based approaches

Disassembly-based approaches use assembly code to create features. Our study is categorized as this method. Yan *et al.*[2] proposed MAGIC for classifying control flow graphs using DGCNN[15]. MAGIC manually designs basic block features. Awad *et al.*[16] handle opcode sequences as documents and apply a Word2vec model to obtain malware features. They use k-Nearest Neighbour to classify malware. Ni *et al.*[17] proposed MCSC that transforms opcode sequences into vectors using SimHash[17] and classifies these vectors with convolutional neural networks. In the experiments of Ma *et al.*[6], the disassembly-based method performed relatively worse than the other methods. The MalwareBazaar dataset demonstrates that image-based and binary-based methods achieve average F1-Scores of 96.65% and 97.1%, respectively, while disassembly-based methods yield a lower score of 91.98%. Our method aims to achieve high performance in classifying malware based on function features, even when employing disassembly-based approaches.

Lately, GEMAL, an FCG-based malware classification method, has been proposed by Wu *et al.*[3]. GEMAL converts assembly instructions into vectors using Word2vec and adds them up for each function to create feature vectors of functions. To vectorize an FCG, GEMAL adopts a graph embedding network with an attention mechanism that draws inspiration from the work of Massarelli *et al.*[18]. GEMAL achieved an impressive accuracy of 99.81% in classifying malware families on the BIG-2015 dataset[19] provided by Microsoft, outperforming existing methods. However, it is unclear whether the FCG structure is effective for malware classification only from the results of Wu *et al.*[2,3]. The effect of the FCG structure should be investigated, as it may be possible to classify well using only functional features without using the FCG structure. We propose a new malware classification method that extends the GEMAL method. Our method creates function embedding vectors using SAFE[4], which can reflect the context of the instructions instead of Word2Vec. We then investigate the effect of the FCG structure on malware classification by comparing it with a model that does not use the FCG structure. Due to the unavailability of the GEMAL source code, we implemented the method based on the original paper[3] and utilized the source code[20] by Massarelli *et al.*[18], which GEMAL was derived from. We then conduct replication experiments to compare the outcomes with our approach.

3. PRELIMINARY

3.1. Graph Neural Network

Graph Neural Networks (GNNs)[1] have gained much interest in the field of machine learning in recent years and have been applied to many domains related to graph structures, such as human relationships, molecules, and networks[21]. They can be applied to several tasks, including node classification, link prediction, and graph classification. Our method employs GNNs to classify graphs of FCGs, where program functions are nodes and functions in a calling relationship are linked by edges.

3.1.1. Graph formulation

A directed graph $G = (V, E)$ comprises a collection of nodes $V = \{v_1, ..., v_n\}$ and edges $E = \{e_1, ..., e_n\}$. Each edge $e = (u, v) \in E$ connects a node u to v. The adjacent nodes of a node v are represented by $\mathcal{N}(v) = u \mid (u, v) \in E$. For each node v, an initial feature vector $\mathbf{x}v \in \mathbb{R}^d$ is assigned, where d represents the dimension of the feature space. The graph comprises a total of n nodes, and the feature matrix is denoted as $\mathbf{X} \in \mathbb{R}^{n \times d}$. We use $\mathbf{h}v$ to represent the latent feature vector of a node, while $\mathbf{h}_G$ corresponds to the feature vector of the graph.

3.1.2. Graph classification in GNN

Classifying graphs in GNNs is performed in the following three main steps.

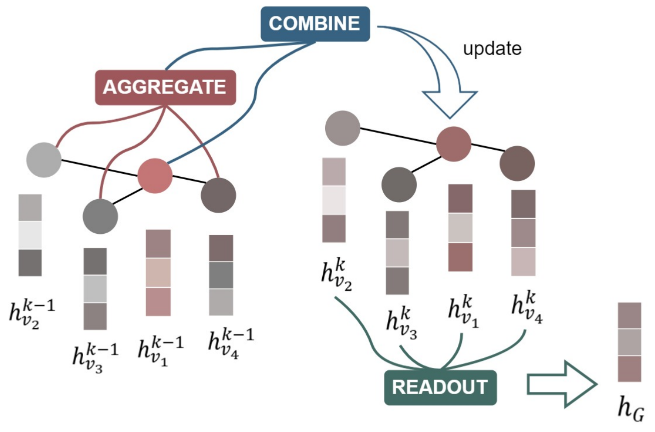


Figure 2. Overview of the process of graph classification in GNN. GNN: Graph Neural Network.

1. The features of adjacent nodes are aggregated using the AGGREGATE function.
2. The features of the node are updated using the aggregated features of its neighbors by the COMBINE function.
3. A feature vector representing the entire graph is obtained from the individual node features using the READOUT function.

The kth updated formula for a node is expressed as follows:

$$\mathbf{h}_v^k \leftarrow \text{COMBINE}\left(\mathbf{h}_v^{k-1}, \text{AGGREGATE}\left(\left\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\right\}\right)\right)$$

To illustrate this process, consider the example shown in Figure 2. The set of neighbors of v_1 is $\{v_2, v_3, v_4\}$. The latent feature vectors of these nodes, $\mathbf{h}{v_2}^{k-1}$, $\mathbf{h}{v_3}^{k-1}$, and $\mathbf{h}{v_4}^{k-1}$, are input to the function AGGREGATE. The output of AGGREGATE, along with the current feature vector $\mathbf{h}_{v_1}^{k-1}$, is passed to the COMBINE function, resulting in an updated feature vector $\mathbf{h}_{v_1}^{k}$. This process is repeated for all nodes in the graph. Finally, READOUT summarizes the feature vectors of all nodes, yielding a single $\mathbf{h}_G$ that represents the entire graph.

Several types of READOUT are proposed, including average, maximum, and sum. This process allows us to obtain $\mathbf{h}_G$ even if the number of nodes differs in each sample.

3.1.3. GraphSAGE

We use GraphSAGE[22], a representative GNN model in our proposed method. Our proposed method uses the Mean Aggregator defined by the following equation.

$$\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W} \cdot \text{MEAN}\left(\left\{\mathbf{h}_v^{k-1}\right\} \cup \left\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\right\}\right)\right)$$

The MEAN function takes a set of vectors as input and outputs an averaged vector. The weight $\mathbf{W}$ is a learning parameter. The average of the feature $\mathbf{h}_v^{k-1}$ and its neighbor nodes $\mathbf{h}_u^{k-1}$ are weighted, and the activation function is applied. This way, the feature vectors of the neighbor nodes are convolved to obtain node vectors that reflect the graph structure.

3.2. SAFE

SAFE[4] is a network proposed by Massarelli *et al.*[4] for generating embedding vectors representing function features. We use SAFE for function embedding. The SAFE architecture is shown in Figure 3 (cited from[4]).

First, the assembly instructions in the function are input to the i2v (instruction to vector) network, which converts the instructions to vectors. i2v acquires embedding vectors corresponding to assembly instructions

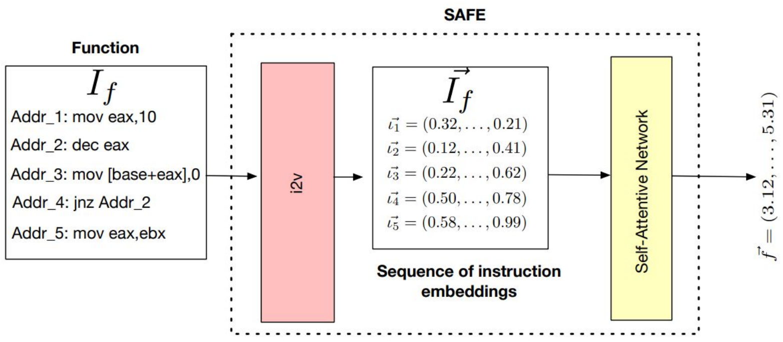

SAFE
Function
I_f
Addr_1: mov eax,10
Addr_2: dec eax
Addr_3: mov [base+eax],0
Addr_4: jnz Addr_2
Addr_5: mov eax,ebx
i2v
$\vec{I_f}$
$\vec{\iota_1} = (0.32, \ldots, 0.21)$
$\vec{\iota_2} = (0.12, \ldots, 0.41)$
$\vec{\iota_3} = (0.22, \ldots, 0.62)$
$\vec{\iota_4} = (0.50, \ldots, 0.78)$
$\vec{\iota_5} = (0.58, \ldots, 0.99)$
Sequence of instruction embeddings
Self-Attentive Network
$\vec{f} = (3.12, \ldots, 5.31)$

Figure 3. Architecture of SAFE (cited from [4]). i2v converts instructions to vectors, and Self-Attentive Network obtains a vector $\vec{f}$ that represents the entire function.

using the skip-gram method[9]. The i2v model is trained by predicting the instructions around the current instruction. Corresponding to natural language, instructions are treated as words and functions as sentences.

The i2v model is trained with filtered instructions as follows:

- Replaces the memory address with "MEM".
- Replaces the number exceeding the threshold value (in this case, 5000) with "IMM".
- Replaces the remaining number with "N".
- Connects operands and opcodes with "_".

Then, the Self-Attentive Network processes the instruction vectors as time-series data using RNN and the Attention mechanism. The important instruction vectors are largely weighted by using the Attention mechanism. This network enables obtaining function features accurately, even for large functions.

Massarelli *et al.*[4] showed that functions are classified into multiple categories (e.g., encryption algorithms, mathematical operations, string operations, etc.) using function embedding vectors generated by SAFE. In other words, SAFE can generate function embedding vectors that reflect the processing contents of the functions.

4. METHODS

Figure 4 shows an overview of the proposed method SAFE+GNN. SAFE+GNN is composed of three parts: creating an FCG, extracting function features, and classifying malware. SAFE+GNN creates FCG and feature vectors of functions from the input binary file. The feature vectors are extracted by SAFE and assigned to nodes in the graph. Finally, malware families are classified by the GNN using the FCG as input. All SAFE+GNN modules are written in Python.

4.1. Preprocessing

SAFE+GNN creates FCGs, extracts function features in the preprocessing phase, and uses a reverse engineering tool, Ghidra[23], for preprocessing. Ghidra has many features, such as disassembling, decompiling, etc. Specifically, SAFE+GNN creates FCGs and extracts function code as binary strings using Ghidra. An FCG is a directed graph that captures the relationship between functions, with nodes representing functions and edges representing function calls. Edges are connected from the calling function to the called function, and features are aggregated in this direction.

SAFE is used for feature extraction of function. We described the details of SAFE in Section 3.2. We use a PyTorch implementation of SAFE[24] and a trained model in AMD64 programs[25]. Here, the assembly

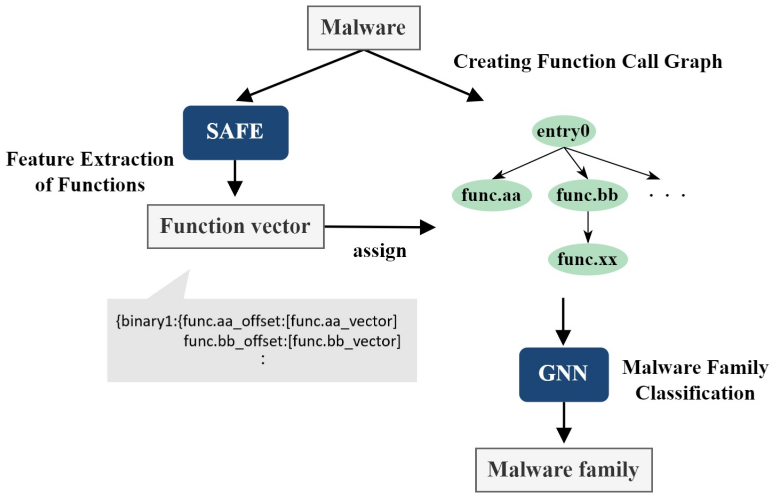

Malware
Creating Function Call Graph
SAFE
Feature Extraction of Functions
Function vector
assign
entry0
func.aa
func.bb
func.xx
{binary1:{func.aa_offset:[func.aa_vector] func.bb_offset:[func.bb_vector] :
GNN
Malware Family Classification
Malware family

Figure 4. Overview of our proposed method SAFE+GNN.

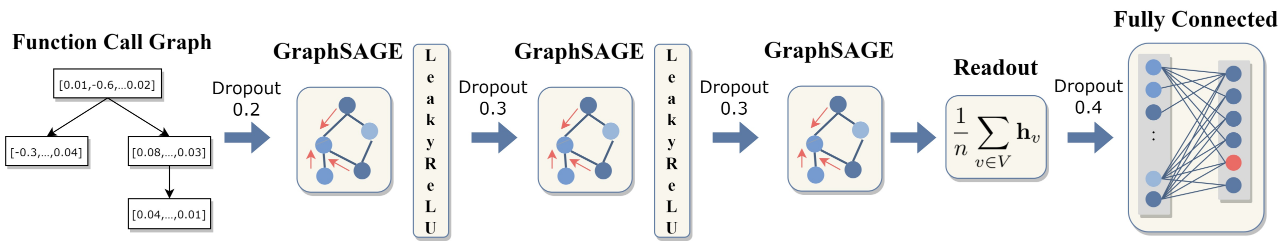


Figure 5. Classification model in SAFE+GNN. This model corresponds to GNN in Figure 4

instruction sequence for each function is converted into a 100-dimensional feature vector. This feature vector is expected to represent the semantics of the function. This module outputs the function offsets and feature vectors from the binary file and assigns them as features of the nodes in the FCG.

4.2. Malware classification model

The classification model was implemented using PyTorch 1.11.0a0+ b6df043 and PyToch Geometric 2.0.2. Figure 5 shows the architecture of the classification model. This model corresponds to the GNN in Figure 4.

This model takes FCGs with 100-dimensional function vectors of node features as input. FCGs are convoluted with GraphSAGE and activated with LeakyReLU, which is shown by

$$\mathbf{h}_v^k \leftarrow \mathrm{LeakyReLU}\left(\mathbf{W}_1 \cdot \mathrm{MEAN}\left(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}\right)\right),$$

where

$$\mathrm{LeakyReLU} = \begin{cases} x & (x > 0) \\ 0.01x & (x <= 0), \end{cases}$$

$\mathbf{W}_1$ is a learning parameter. After three convolutions in GraphSAGE, the readout process is executed to obtain a feature vector for the entire graph. SAFE+GNN uses the readout process by the average value expressed in the following equation:

$$\mathbf{h}_G = \frac{1}{n}\sum_{v\in V}\mathbf{h}_v$$

Table 1. Sample Counts(MalwareBazaar)

Family	Number of Samples
Gozi	763
GuLoader	564
Heodo	209
IcedID	577
njrat	942
Trickbot	872
Total	3,927

Table 2. Sample Counts(BIG-2015)

Family	Number of Samples
Ramnit	1,533
Lollipop	2,475
Kelihos_ver3	2,938
Vundo	454
Simda	42
Tracur	751
Kelihos_ver1	387
Obfuscator.ACY	1,217
Gatak	1,013
Total	10,810

In malware classification, the number of nodes (i.e., the number of functions) is usually different for each sample, but the readout process can accommodate variable nodes. Classification is then performed in the Fully Connected layer, which takes a graph vector $\mathbf{h}_G$ as input and outputs a vector that represents the classification probabilities of classes (families) as follows:

$$\mathbf{p} = \mathrm{softmax}(\mathbf{W}_2 \cdot \mathbf{h}_G)$$

where $\mathbf{W}_2$ is a learning parameter. The elements of $\mathbf{p}$ represent the probability that each class is predicted. That is, the class with the highest predicted probability represents the predicted class.

Dropout[26] is used to mitigate overfitting and stabilize learning. Dropout randomly sets the output of a layer to 0 during training to be correctly recognized even if some data are dropped. This improves the robustness of the model by avoiding overfitting of some local features. In Figure 5, Dropout 0.2 means that the output of the node is set to 0 with a probability of 0.2.

5. RESULTS

In this experiment, we conduct evaluation experiments with two classification models: SAFE+GNN of our proposed method and SAFE+MLP, which ignores the graph structure to check the effect of that. Specifically, SAFE+MLP replaces the GraphSAGE layer of SAFE+GNN with a fully connected layer, and the other conditions are the same. Note that SAFE+MLP does not use graph edges. We examine the effects of FCG structure on malware classification by comparing the experimental results of two models. We also implemented and experimented with GEMAL, described in Section 2.3 as a comparison.

5.1. Datasets and preprocessing

We used the two datasets used in the experiments of the existing study by Ma *et al.* Using SAFE+GNN, we created function vectors and FCGs for each dataset. The function vectors were then assigned to the nodes of the FCGs to create a graph-format dataset. The datasets we used are as follows.

5.1.1. MalwareBazaar dataset

The first is the MalwareBazaar dataset, created by Ma *et al.*[6]. MalwareBazaar[27] is a service that makes malware specimens available for research. This dataset was created by the following procedures. First, the top six malware families released in MalwareBazaar during 2020 were selected, and 1000 malware samples were downloaded for each of these families. Next, non-PE format samples were excluded. The labels of each sample were verified using Joe Security[28] and AVClass[29], and samples with inconsistent labels were removed. As a result, a dataset containing 3,971 samples from 6 families was created. Ma *et al.* have released a list of sample hashes and their families on GitHub[30]. We used the list and the API of MalwareBazaar and downloaded 3,971 malware samples. Among them, we used the 3,927 samples that were successfully preprocessed (Table 1) for the experiments.

5.1.2. BIG-2015 dataset

The second dataset is known as BIG-2015, which was released by Microsoft and contains 10,868 samples representing nine different families. This dataset contains two formats: .byte files, which are binary representations of hexadecimal numbers, and .asm files, which are the disassembled output of IDA Pro. To conduct the experiment, we generated FCGs from the .asm files and extracted the function binaries. For the evaluation, a total of 10,810 samples (as shown in Table 2) were successfully preprocessed and used.

5.2. Model training and evaluation

We apply a ten-fold cross-validation method to evaluate the performance of SAFE+GNN and SAFE+MLP. This experiment was performed using a single NVIDIA A100 GPU. The parameter settings during training are as follows.

- Learning rate: 0.005
- Mini batch size: 128
- Number of epochs: 700
- Hidden layer dimensions: 64

We use the cross-entropy loss as the loss function. When training the model, the learning parameters are adjusted to minimize the output of the following equation:

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log\left(p_{i,c}\right)$$

where N is the number of observations in the dataset, C is the number of malware families in the dataset, $y_{i,c}$ is 1 if the i th sample belongs to malware family c, and $p_{i,c}$ is the predicted probability that i th sample is in family c in the output of the model.

AdamW[31] is used as the optimization algorithm and is a modification of Adam[32] that allows regularization to work. Regularization has the effect of preventing overfitting. In experiments using the MalwareBazaar dataset, it took an average of 55.8 minutes to train 700 epochs.

5.3. Evaluation metrics

To compare with the experiment of Ma *et al.*[6], we use the same evaluation metrics as theirs. We use the following four evaluation metrics: Accuracy, Precision (P_{macro}), Recall (R_{macro}), and F1-Score ($F1_{\text{macro}}$). Each formula is expressed by

$$Accuracy = \left(\sum_{n=1}^{N}\sum_{m=1}^{M} TP_{mn}\right)/S \qquad P_{\text{macro}} = \left(\sum_{n=1}^{N} P_n\right)/N$$

$$R_{\text{macro}} = \left(\sum_{n=1}^{N} R_n\right)/N \qquad F1_{\text{macro}} = \left(\sum_{n=1}^{N} F1_n\right)/N,$$

where

$$P_n = \left(\sum_{m=1}^{M} TP_{mn}\right)/\left(\sum_{m=1}^{M}(TP_{mn}+FP_{mn})\right) \qquad R_n = \left(\sum_{m=1}^{M} TP_{mn}\right)/\left(\sum_{m=1}^{M}(TP_{mn}+FN_{mn})\right)$$

$$F1_n = (2*P_n*R_n)/(P_n+R_n),$$

N is the number of all classes, M is the number of cross-validations, and S is the number of all samples. TP_{mn}, FP_{mn}, and FN_{mn} represent the true positive, false positive, and false negative of malware family n in the m-fold.

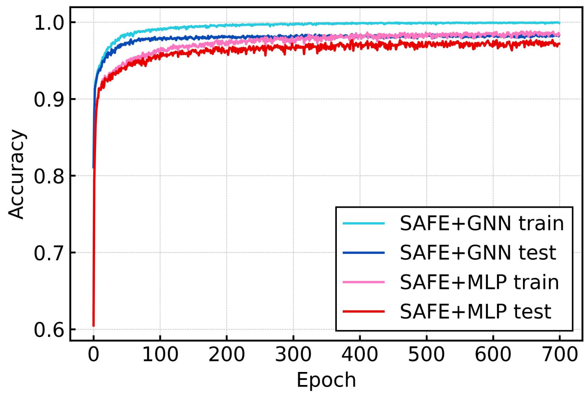


Figure 6. Learning curve of accuracy. SAE+GNN learns faster and has a higher average accuracy than SAFE+MLP.

Table 3. Results of classification performance evaluation experiments

Category	Model	MalwareBazaar dataset				BIG-2015			
		Accuracy	P_{macro}	R_{macro}	$F1_{macro}$	Accuracy	P_{macro}	R_{macro}	$F1_{macro}$
Binary	CBOW+MLP [8] *	**97.81**	**97.92**	**98.08**	**98.00**	98.41	97.63	96.67	97.12
	MalConv [7] *	95.92	96.04	96.43	96.20	97.02	94.34	92.62	93.33
Image	ResNet-50 [12] *	96.68	96.91	96.75	96.83	98.42	96.57	95.68	96.08
	VGG-16 [14] *	96.35	96.58	96.54	96.56	93.94	90.32	81.89	87.27
	Inception-V3 [11] *	95.83	95.67	95.79	95.73	96.99	93.67	94.46	94.03
	IMCFN [13] *	97.38	97.53	97.41	97.47	97.77	95.93	94.81	95.13
Disassembly	MAGIC [2] *	92.82	88.03	87.36	87.45	98.05	96.75	94.03	95.14
	Word2vec+KNN [16] *	95.64	93.34	94.29	93.79	98.07	96.41	96.51	96.45
	MCSC [17] *	96.80	94.97	94.51	94.70	97.94	95.97	96.17	96.06
	SAFE+GNN(proposed method)	**98.32**	**98.06**	**98.49**	**98.27**	**99.10**	**98.67**	**97.98**	**98.31**
	SAFE+MLP(for comparison)	97.20	96.75	97.28	97.00	98.94	98.21	97.07	97.60
	GEMAL(replication) [3]	97.71	97.65	98.00	97.82	**99.37**	**98.26**	**98.48**	**98.37**
	GEMAL(paper) [3] †	-	-	-	-	99.81	-	-	99.81

* Demonstration results by Ma *et al.* [6].
† Experimental results by Wu *et al.* [3].
The top two scores in each evaluation metric are shown in bold.

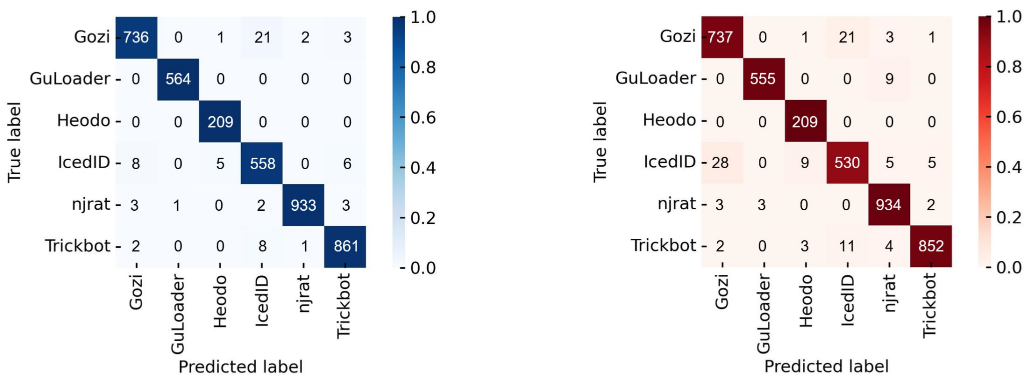
5.4. Experimental results

The learning curves for the average accuracy in the ten-fold cross-validation of SAFE+GNN and SAFE+MLP are shown in Figure 6. At the end of 700 epochs, the learning curves for both models are almost flat on the training data, indicating that the learning has converged. Compared to SAFE+MLP, SAFE+GNN tends to have higher average accuracy and learn faster.

Table 3 shows the classification performance and compares them with the existing studies. For GEMAL, we evaluate performance using scores of replicated experiments instead of paper scores to compare also for the MalwareBazaar dataset. On the MalwareBazaar dataset, SAFE+GNN outperformed all other methods on all evaluation metrics. Comparing the F1-scores of SAFE+GNN and SAFE+MLP, SAFE+GNN is 1.27% higher. On BIG-2015, the F1-score of SAFE+GNN is 98.31%, which is approximately equivalent to 98.37% of GEMAL. Accuracy is 99.37% for GEMAL versus 99.10% for SAFE+GNN. However the standard deviation in the 10-fold cross-validation of SAFE+GNN was 0.70%, so the performance difference is within the margin of error and considered almost equivalent to GEMAL.

The confusion matrixes of SAFE+GNN and SAFE+MLP (Figure 7) show that SAFE+GNN slightly improves classification performance, especially for the IcedID and GuLoader samples.

6. DISCUSSION



(a) Confusion matrix of SAFE+GNN. **(b)** Confusion matrix of SAFE+MLP.

Figure 7. Confusion matrixes in experiments with the MalwareBazaar dataset. The total predictions of the test data for ten-fold cross-validation are plotted. The colors in the heatmaps are normalized over the true conditions. The numbers displayed in the cells are before normalization.

6.1. Effectiveness of our method

The results of the evaluation experiments show that SAFE+GNN is capable of classifying malware with world-leading performance. SAFE+GNN and GEMAL are outstanding among disassembly-based methods, and SAFE+MLP also outperformed them. It is assumed that function features improved classification performance since there is no method using function features in the experiments of Ma *et al.*

In the experiments of Ma *et al.* , the performance of disassembly-based methods is poor, especially on the MalwareBazaar dataset. This lower performance can be attributed to the lack of PE header features in these methods. Existing research has indicated that models that take the entire executable file into account, such as MalConv[7], primarily focus on the PE header during malware classification[33]. However, it is risky to classify malware by relying on the PE header as it contains spoofable data (e.g., timestamps). SAFE+GNN outperformed methods that use the whole file, although it uses features of code segments only. SAFE+GNN is considered to more accurately represent malware features by using function as a feature.

6.2. Investigation of the effects of Function Call Graph structure

We investigate the relationship between graph structures and classification results in the MalwareBazaar dataset to consider samples GNN is good at. We focus on the number of nodes to generalize the analysis to GNN-based methods. We define same-num_nodes-group as a group of samples with the same number of nodes and will proceed with the discussion assuming that same-num_nodes-group in the same family contains similar samples. The number of nodes corresponds to the number of functions in the sample.

Figure 8 shows accuracy calculated for each same-num_nodes-group. In other words, it shows how the performance varies with the number of similar samples. In the groups with many similar samples (more than 100), both SAFE+GNN and SAFE+MLP show good performance and almost no misclassification. On the other hand, in the groups without many similar samples (less than 15), the performance difference between SAFE+GNN and SAFE+MLP is observed. SAFE+GNN tends to perform relatively well even when there are not many similar samples.

In Figure 9, sample counts against the number of nodes are plotted for each family, and the colors reflect the classification results for SAFE+GNN and SAFE+MLP. Note that if the blue area (SAFE+GNN only correct) is larger than the red area (SAFE+MLP only correct), we can read SAFE+GNN is superior to SAFE+MLP. We can see the difference in the classification results of both models depending on the number of nodes and the number of similar samples (corresponding to sample counts).

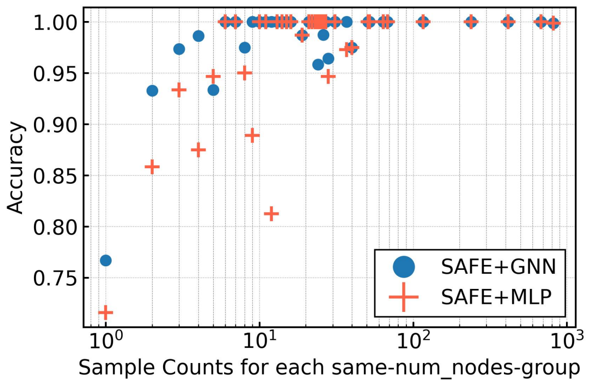


Figure 8. Accuracy calculated for each same-num_nodes-group. The horizontal axis is a logarithmic scale. same-num_nodes-group means a group of samples with the same number of nodes.

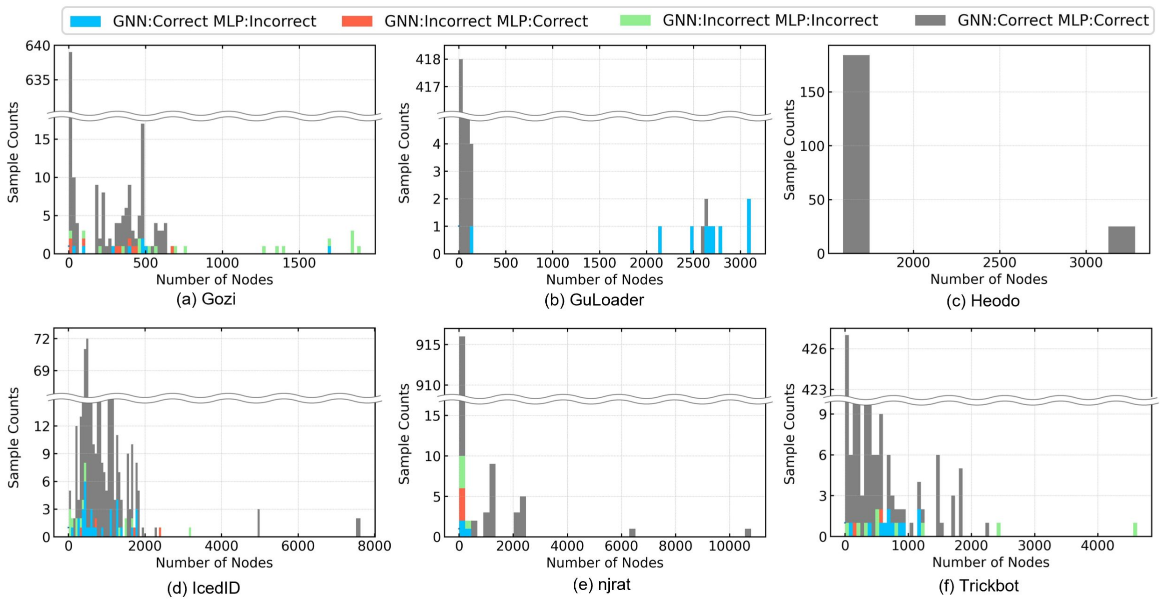


Figure 9. Sample counts against the number of nodes. The vertical axis is partially cut off by a wavy line. Colors reflect the classification results for SAFE+GNN and SAFE+MLP.

For Heodo, both SAFE+GNN and SAFE+MLP successfully classified with 100% performance. It can be seen from Figure 9(c) that the number of nodes in the Heodo samples has only two values. It is possibly inferred that the samples with the same number of nodes are so close in the feature space that they can be accurately classified. We consider that these similar samples with the same number of nodes may differ only in the config information, such as the destination address. For Gozi and njrat, both models have overall similar classification performance, although there are differences in the classification results for each sample. On the other hand, GuLoader, IcedID, and Trickbot stand out with samples incorrectly classified in SAFE+MLP but correctly classified in SAFE+GNN. In particular, SAFE+GNN successfully classified all nine samples in GuLoader that failed to be classified by SAFE+MLP. The GuLoader samples are broadly divided into two groups: one with a small number of nodes and the other with a large number of nodes [Figure 9(b)]. The latter group obviously has fewer similar samples. Most of the samples that were successfully classified only for SAFE+GNN belong to the group with a large number of nodes.

In order to further investigate the GuLoader samples, we used PEiD[34] to identify the development languages. As a result, 552 of the 555 samples successfully classified by SAFE+GNN and SAFE+MLP were identified as written with Microsoft Visual Basic 5.0/6.0. On the other hand, eight of the nine samples that failed to be classified by SAFE+MLP were identified as written with Borland Delphi 6.0-7.0. In this way, even within the same malware family, there are samples written with different development languages, and their characteristics differ significantly. SAFE+GNN accurately classified a small number of similar sample groups for the GuLoader samples. We consider the result is the effect of the GNN, which incorporates the features of the called function and updates them to effective features for malware classification. On the other hand, SAFE+MLP cannot incorporate features from other functions. Thus, its representation is limited.

6.3. Limitations

As with existing static analysis methods, SAFE+GNN has difficulty performing malware classification on obfuscated malware, such as packed programs. A packed program is implemented with a decompression codes corresponding to each packer. Therefore, the proposed method can be used to identify the characteristics of the packer and is likely to be useful for the packer estimation.

Our proposed method has difficulty in classifying completely new malware correctly because the classification is based on similar malware. In reality, however, we rarely see completely new malware, and even new species generally share some modules with other malware. This method focuses on functions, so it is possible to classify malware based on the modules that they share.

To adapt the model to real-world situations, the model needs to be trained on a larger dataset. The malware used in this experiment exceeded 14,000 malware samples, which is not a small size, but we believe it can be made practical by further increasing the number of sample categories.

7. CONCLUSIONS

In this paper, we proposed SAFE+GNN, a malware classification method based on FCGs and function embedding. The evaluation experiments showed that SAFE+GNN achieved higher classification performance than existing methods. The results indicate the potential power of using function features and FCG structure in malware classification. Furthermore, we discussed the effectiveness of GNNs in malware classification by comparing SAFE+GNN and SAFE+MLP, which ignores the FCG structure. The results showed that SAFE+GNN has an advantage, especially for samples with a small number of similar samples in the dataset. We believe these advantages of SEFE+GNN can be adapted to real-world malware classification since malware changes with the times.

Future experiments should be conducted on large datasets closer to real-world conditions. It is also necessary to handle packers and other obfuscation.

DECLARATIONS

Authors' contributions

Made substantial contributions to the conception and design of the study and performed data analysis and interpretation: Someya M, Otsubo Y, Otsuka A

Availability of data and materials

The research artifacts are available at the following URL: https://github.com/som3ya/SAFE-GNN

Financial support and sponsorship

None.

Conflicts of interest
All authors declared that there are no conflicts of interest.

Ethical approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Copyright

REFERENCES

1. Scarselli F, Gori M, Ah Chung Tsoi, Hagenbuchner M, Monfardini G. The graph neural network model. *IEEE Trans Neural Netw* 2009;20:61-80. DOI
2. Yan J, Yan G, Jin D. Classifying malware represented as control flow graphs using deep graph convolutional neural network. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN); 2019. p. 52-63. DOI
3. Wu XW, Wang Y, Fang Y, Jia P. Embedding vector generation based on function call graph for effective malware detection and classification. *Neural Comput Applic* 2022;34:8643-56. DOI
4. Massarelli L, Di Luna GA, Petroni F, Querzoni L, Baldoni R. Function Representations for Binary Similarity. *IEEE Trans Dependable and Secure Comput* 2022;19:2259-73. DOI
5. Ding SHH, Fung BCM, Charland P. Asm2Vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In: 2019 IEEE Symposium on Security and Privacy (SP); 2019. p. 472-89. DOI
6. Ma Y, Liu S, Jiang J, Chen G, Li K. A comprehensive study on learning-based PE malware family classification methods. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering; 2021. p. 1314-25. DOI
7. Raff E, Barker J, Sylvester J, et al. Malware Detection by Eating a Whole EXE. In: The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence, 2018. vol. WS-18 of AAAI Technical Report. AAAI Press; 2018. p. 268-76. Available from: https://aaai.org/ocs/index.php/WS/AAAIW18/paper/view/16422. [Last accessed on 5 Jul 2023]
8. Qiao Y, Zhang B, Zhang W. Malware classification method based on word vector of bytes and multilayer perception. In: ICC 2020-2020 IEEE International Conference on Communications (ICC); 2020. p. 1-6. DOI
9. Mikolov T, Sutskever I, Chen K, Corrado G, Dean J. Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2. NIPS'13. Red Hook, NY, USA: Curran Associates Inc; 2013. p. 3111-9. Available from: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html. [Last accessed on 5 Jul 2023]
10. Nataraj L, Karthikeyan S, Jacob G, Manjunath BS. Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security - VizSec '11. Pittsburgh, Pennsylvania: ACM Press; 2011. p. 1-7. DOI
11. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016. p. 2818-26. DOI
12. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016. P. 770-78. DOI
13. Vasan D, Alazab M, Wassan S, et al. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Comput netw* 2020;171:107138. DOI
14. Simonyan K, Zisserman A. Very deep convolutional networks for Large-scale Image Recognition. In: 3rd International Conference on Learning Representations, ICLR 2015; 2015. Available from: http://arxiv.org/abs/1409.1556. [Last accessed on 5 Jul 2023]
15. Zhang M, Cui Z, Neumann M, Chen Y. An end-to-end deep learning architecture for graph classification. In: AAAI Conference on Artificial Intelligence; 2018. DOI
16. Awad Y, Nassar M, Safa H. Modeling malware as a language. In: 2018 IEEE International Conference on Communications (ICC); 2018. p. 1-6. DOI
17. Ni S, Qian Q, Zhang R. Malware identification using visualization images and deep learning. *Computers & Security* 2018;77:871-85. DOI
18. Massarelli L, Di Luna GA, Petroni F, Querzoni L, Baldoni R. Investigating graph embedding neural networks with unsupervised features extraction for binary analysis. Proceedings 2019 Workshop on Binary Analysis Research; 2019. DOI
19. Ronen R, Radu M, Feuerstein C, Yom-Tov E, Ahmadi M. Microsoft malware classification challenge. ArXiv 2018 Feb. Available from: http://arxiv.org/abs/1802.10135. [Last accessed on 5 Jul 2023]
20. Massarelli L. Unsupervised-features-learning-for-binary-similarity: code for the paper "Investigating graph embedding neural networks with unsupervised features extraction for binary analysis";. Accessed: 2022-12-24. https://github.com/lucamassarelli/Unsupervised-Fea

tures-Learning-For-Binary-Similarity. [Last accessed on 5 Jul 2023]
21. Wu Z, Pan S, Chen F, et al. A comprehensive survey on graph neural networks. *IEEE Trans Neural Netw Learn Syst* 2021;32:4-24. DOI
22. Hamilton WL, Ying Z, Leskovec J. Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017; 2017. p. 1024-34. Available from: https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html. [Last accessed on 5 Jul 2023]
23. Ghidra. Accessed: 2022-12-24. https://ghidra-sre.org/. [Last accessed on 5 Jul 2023]
24. SAFEtorch: pytorch version of the SAFE neural network. Accessed: 2022-12-24. https://github.com/facebookresearch/SAFEtorch. [Last accessed on 5 Jul 2023]
25. SAFEtorch-model. Accessed: 2022-12-24. http://dl.fbaipublicfiles.com/SAFEtorch/model.tar.gz. [Last accessed on 5 Jul 2023]
26. Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. *J Mach Learn Res* 2014;15:1929-58. DOI
27. MalwareBazaar. Accessed: 2022-12-24. https://bazaar.abuse.ch/. [Last accessed on 5 Jul 2023]
28. Joe Security LLC. Deep malware analysis-Joe Sandbox. Accessed: 2022-12-24. https://www.joesecurity.org/. [Last accessed on 5 Jul 2023]
29. Sebastián M, Rivera R, Kotzias P, Caballero J. AVclass: a tool for massive malware labeling. In: vol. 9854. Cham; 2016. p. 230-53. DOI
30. Benchmarking-Malware-Family-Classification. Accessed: 2022-12-24. https://github.com/MHunt-er/Benchmarking-Malware-Family-Classification/. [Last accessed on 5 Jul 2023]
31. Loshchilov I, Hutter F. Decoupled weight decay regularization. In: 7th International Conference on Learning Representations, ICLR 2019; 2019. Available from: https://openreview.net/forum?id=Bkg6RiCqY7. [Last accessed on 5 Jul 2023]
32. Kingma DP, Ba J. Adam: a method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015; 2015. Available from: http://arxiv.org/abs/1412.6980. [Last accessed on 5 Jul 2023]
33. Bose S, Barao T, Liu X. Explaining AI for malware detection: analysis of mechanisms of MalConv. In: 2020 International Joint Conference on Neural Networks (IJCNN). IEEE; 2020. DOI
34. PEiD - aldeid. Accessed: 2022-12-24. https://www.aldeid.com/wiki/PEiD. [Last accessed on 5 Jul 2023]