# Malware Self-Supervised Graph Contrastive Learning with Data Augmentation

Yun Gao*, Hirokazu Hasegawa†, Yukiko Yamaguchi*, and Hajime Shimada*
*Information Technology Center
Nagoya University, Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan
Email: gaoyun@net.itc.nagoya-u.ac.jp, yamaguchi@itc.nagoya-u.ac.jp, shimada@itc.nagoya-u.ac.jp
†Center for Strategic Cyber Resilience
National Institute of Informatics, Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan
Email: hasegawa@nii.ac.jp

***Abstract*—Traditional malware detection methods struggle to quickly and effectively keep up with the massive amount of newly created malware. Based on the features of samples, machine learning is a promising method for the detection and classification of large-scale, newly created malware. The current research trend uses machine-learning technologies to rapidly and accurately learn newly created malware. In this paper, we propose a malware classification framework based on Graph Contrastive Learning (GraphCL) with data augmentation. We first extract the Control-Flow Graph (CFG) from portable executable (PE) files and simultaneously generate node feature vectors from the disassembly code of each basic block through MiniLM, a large-scale pre-trained language model. Then four different data augmentation methods are used to expand the graph data, and the final graph representation is generated by the GraphCL model. These representations can be directly applied to downstream tasks. For our classification task, we use C-Support Vector Classification (SVC) as a classification model. To evaluate our approach, we made a CFG-based malware classification dataset from the PE files of the BODMAS Malware Dataset, which we call the Malware Geometric Multi-Class Dataset (MGD-MULTI), and collected the results. In addition, we added a new public malware graph dataset called MALNET-TINY. We also employed Local Degree Profile (LDP) to generate node representations for the graph data. The evaluation results on two malware graph dataset show that our proposal has great potential. According to our experimental evaluation, the self-supervised learning method can also achieve superior results, even surpassing the supervised learning method.**

***Keywords*—malware classification; graph contrastive learning; data augmentation; self-supervised learning.**

## I. Introduction

In our previous work [1], we implemented self-supervised representation learning on the self-built MGD-Multi dataset, and used the representation to carry out the downstream malware classification task. In this work, we added a new public dataset to further verify the effectiveness of self-supervised contrastive learning, and evaluated different data augmentation combination.

Fueled by the progress of software technology and the internet's development, thousands of malware are created every day due to the proliferation of malware creation and obfuscation tools. Such a massive flood of data poses a considerable challenge to malware analysts and security response centers (SOCs). Traditional malware detection methods cannot continue to quickly and effectively detect such a massive amount of newly created malware. In past decades, machine learning has played an important role in information security, especially in malware detection and classification tasks. It is also a promising method to detect and classify large-scale newly created malware using the features of samples.

In the field of static malware detection, the feature extraction method of portable executable (PE) files used in the Endgame Malware Benchmark for Research (EMBER) dataset [2] has been widely applied. This feature extraction method directly provides consistent feature vectors to researchers, allowing individuals in the same field to compare their respective proposed methods. The information related to software structure, such as CFG, is rarely extracted, and most methods are based on surface analysis for extracting statistical information as features. In addition, in most malware detection and classification scenarios, the model is supervised for end-to-end training.

Supervised learning requires manual labeling of a large amount of data, and the model effect depends on the quality of the labels. Therefore, the future research trend, which is exploring unsupervised learning methods, is critical for malware detection and classification. In recent years, Graph Neural Networks (GNNs) have made remarkable progress. We can exploit their powerful representation ability to better represent malware and improve the effectiveness of its detection and classification. However, one remaining difficulty is how to represent malware in graphical form. CFG is a natural graph structure, we can generate the graph structure data of malware by extracting CFG. Therefore, we seek to classify malware by constructing a graph dataset and using unsupervised learning. Since no publicly available graph classification dataset exists for malware classification, we started by creating such a dataset.

Our contributions can be summarized as follows:

- We propose a malware classification framework based on graph contrastive learning under self-supervised learning.
- We retain the structural information of the samples extracted from CFG and embed the text features of each node with a pre-trained language model.
- We create a special graph dataset for malware classification that can be used directly on GNNs.
- Our pre-trained model can effectively represent malware in low dimensions, and this representation can be used for various downstream tasks.
- We have achieved promising results in the classification task of two malware graph datasets, and compared the effectiveness of different methods.

The remainder of this paper is organized as follows. Section II reviews related researches and highlights their methodological differences. In Section III, we describe the background knowledge of our research, as well as the methods of graph feature extraction and graph data construction. In Section IV, we discuss the principles of our proposed data augmented GraphCL-based static malware PE classification system and its application to malware classification. In Section V, we briefly discuss the implementation details of our proposal. In Section VI, we describe the corresponding experiments and evaluate their feasibility as well as the advantages and limitations of our proposal. Finally, we discuss our conclusion and describe future work in Section VII.

## II. Related Work

Static malware detection allows a sample to be classified as malicious or benign without executing it. In contrast, dynamic malware detection is based on its runtime behavior and as well as its analysis, including time-dependent system call sequences [3]–[5]. A program that precisely discerns a virus from any other program by examining its appearance is infeasible, so static detection is not generally deterministic [6]. Compared with dynamic detection, the advantages of static detection are also obvious. Static detection can identify malicious files before the samples are executed. Since 1995, various machine-learning-based methods for static PE malware detection have been proposed [7]–[9].

### A. Supervised-learning-based Methods

Saxe used histograms through byte-entropy values as input features and multilayer neural networks for classification [8]. Raff et al. showed that fully connected and recursive networks can be applied to malware detection problems [10]. They also used the raw bytes of PE files and built end-to-end deep learning networks [9]. Chen proposed robust PDF malware classifiers with verifiable robustness properties [11]. Coull explored malware detection byte-based deep neural network models to learn more about malware and examined the learned features at multiple levels of resolution, from individual byte embeddings to the end-to-end analysis of models [12]. Rudd proposed ALOHA, which uses multiple additional optimization objectives to enhance the model, including multi-source malicious/benign loss, count loss on multi-source detections, and semantic malware attribute tag loss [13].

### B. Unsupervised-learning-based Methods

Yang presented a novel system called CADE, which can detect drifting samples that deviate from existing classes, and explained detected drift [14].

### C. Graph Representation Learning

The goal of graph representation learning is to preserve as much topological information as possible when mapping nodes into vector representations. We can use its to represent malware, and do the downstream malware classification task.

Graph classification assigns a label to each graph to map the graph to the vector space. A graph kernel is dominant in history. It uses the kernel function to measure the similarity between graph pairs and maps graphs to a vector space with a mapping function. In the context of graph classification, GNNs often employ readout operations to obtain a compact representation at the graph level. GNNs have attracted a lot of attention and demonstrated amazing results in this task.

*1) Supervised Learning:* The Dynamic Graph Convolutional Neural Network [15] (DGCNN) uses K nearest neighbors (KNN), builds a subgraph for each node based on the node's features, and applies a graph convolution to the reconstructed graph. The Graph Isomorphism Network (GIN) [16] presents a GIN that adjusts the weights of the central nodes by learning, theoretically analyzes the GIN's expressiveness better than such GNN structures as the Graph Convolutional Network (GCN), and achieves state-of-the-art accuracy on multiple tasks.

*2) Unsupervised Learning:* Graph2vec [17] uses a set of all the rooted subgraphs around each node as its vocabulary through a skip-gram training process. Infograph [18] applies contrastive learning to graph learning, which is carried out in an unsupervised manner by maximizing the mutual information between graph-level and node-level representations.

*3) Self-Supervised Learning:* Recently, graph contrastive learning has received much attention. It has also been applied in the field of malware detection and classification. EVOLIoT [19] is a novel approach that combats "concept drift" and the limitations of inter-family IoT malware classification by detecting drifting IoT malware families and examining their diverse evolutionary trajectories. This robust and effective contrastive method learns and compares semantically meaningful representations of IoT malware binaries and codes without expensive target labels. Graph contrastive learning (GraphCL) [20] framework for learning self-supervised representations of graph data. GraphCL design four types of graph augmentations to incorporate various priors. GraphMAE [21] is a generative self-supervised graph learning method, which achieves competitive or better performance than existing contrastive methods on tasks including node classification, graph classification, and molecular property prediction. Wiener Graph Deconvolutional Network (WGDN) [22], an augmentation-adaptive decoder empowered by graph wiener filter to perform information reconstruction.

## III. Background

### A. Raw Graph Generation

Most malware samples exhibit polymorphic characteristics, indicating that even slight alterations in the original malware's source code can lead to substantially different compiled code [23], [24]. Cybercriminals often exploit this phenomenon to evade signature-based detection, which is a prevalent method used for malware detection [25]. Fortunately, these subtle source code changes have minimal effect on the CFG and FCG of the executable.

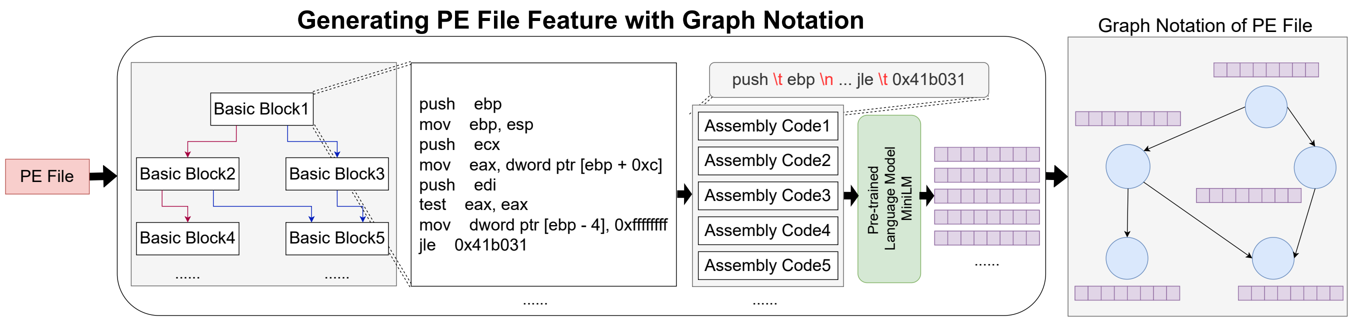

Generating PE File Feature with Graph Notation
Graph Notation of PE File
PE File
Basic Block1
Basic Block2
Basic Block3
Basic Block4
Basic Block5
push ebp
mov ebp, esp
push ecx
mov eax, dword ptr [ebp + 0xc]
push edi
test eax, eax
mov dword ptr [ebp - 4], 0xffffffff
jle 0x41b031
push \t ebp \n ... jle \t 0x41b031
Assembly Code1
Assembly Code2
Assembly Code3
Assembly Code4
Assembly Code5
Pre-trained Language Model MiniLM

Figure 1. Raw Graph Generation for Proposal

*1) CFG Structure and Disassembly Code:* The CFG information is extracted from the original PE file samples, the structure information of the basic blocks is retained, and the disassembly code of each basic block is extracted. Each basic block of CFG has a corresponding disassembly code, and the relationship between each basic block is directional. Disassembly codes need to be transformed into feature vectors of specific dimensions to train GNNs. Malware CFG is usually a very large graph, so it is very time-consuming to extract CFG. Since the disassembly code in each basic block of CFG contains rich semantic information, we need to completely exploit that information and suitably embed it, for example, using a large pre-trained language model.

To train the GNNs, we need to produce graph datasets, and the main task of this module is to convert PE files into raw graphs. The overview of raw graph generation is shown in Figure 1.

*2) FCG Structure and Function Call:* A Function Call Graph (FCG) is a type of graph representation that captures the relationships between functions in a software program or system. It is commonly used in software analysis and program understanding to model how functions in a program interact with each other through function calls. In an FCG, each function is represented as a node, and the edges between nodes indicate function calls. If function A calls function B, there will be a directed edge from node A to node B in the FCG. The FCG provides a visual and structural representation of the flow of control and data between functions within the program.

CFG nodes contain disassembly code, while FCG nodes only include API function names or the starting memory addresses of functions. Although FCG node information is less abundant compared to CFG, it tends to focus more on modeling the graph's structural aspects. It can even generate node features without relying on the existing node information by using graph centrality measures. As a result, compared to CFG, it can efficiently and swiftly extract the graph information from binary files.

## B. Non-attributed Graph Classification

Although the CFG and FCG extracted from binary files contain node information, the extraction of node features and embeddings is a matter that requires consideration. Furthermore, it is necessary to consider whether these node functions are effective.

For the task of graph classification, we need to generate node feature vectors for the graph data. For CFGs, basic blocks contain abundant disassembly code, which can be leveraged to generate node feature vectors. However, for FCGs, each node represents a function call, and there is limited semantic information available, often only comprising API names or function memory addresses. Therefore, we can directly use graph centrality measures and the structural information of the graph itself to generate corresponding node feature vectors.

Below, we introduce two types of node feature generation methods that we use.

*1) Pre-trained Language Model MiniLM:* MiniLM is a method released by Microsoft based on reducing large-scale transformer pre-trained models into smaller models [26]. This Deep Self-Attention Distillation (DSAD) method uses large-scale data for pre-training. The model we use is called "all-MiniLM-L12-v2," which has a 1-billion-sized training set and is designed as a general-purpose model. MiniLM model is a 12-layer transformer with a 384 hidden size and 12 attention heads that contain about 33 M parameters. It maps sentences and paragraphs to a 384-dimensional dense vector space and can be used for tasks like clustering or semantic search. This model is the fastest generation of related studies and still provides good quality. In this step, a 384-dimensional dense vector is generated for each CFG node using the pre-trained model. This vector is added to the corresponding nodes of the directed graph to generate complete graph data with node feature vectors. These directed graphs are used as our raw graph data.

*2) Graph Centrality:* Graph centrality refers to a set of measures used in graph theory and network analysis to determine the importance or influence of individual vertices (nodes) within a graph. Centrality measures aim to identify nodes that play crucial roles in the network's structure, functioning, and communication. Several centrality measures exist, each capturing different aspects of a node's importance. Some of the most common centrality measures include:

*a) Degree Centrality:* Degree centrality is the simplest centrality measure, and it is based on the number of edges incident to a node (its degree). Nodes with higher degrees are considered more central as they have more connections in the network.

*b) Eigenvector Centrality:* Eigenvector centrality measures a node's importance based on the centrality of its

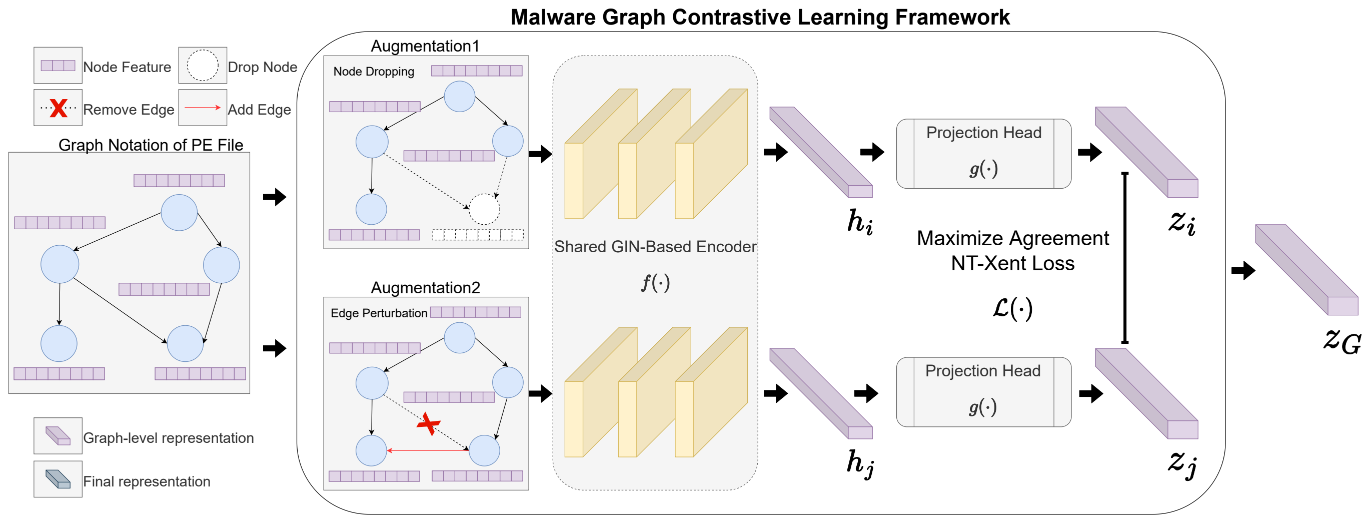


Figure 2. Proposed Malware Graph Contrastive Learning Framework for Graph Representation Generation

neighbors. Nodes connected to other central nodes have higher eigenvector centrality themselves.

*c) PageRank:* Originally developed by Google, PageRank is a variant of eigenvector centrality and is used to rank web pages in search engine results. It assigns a score to each node based on the number and quality of incoming links.

*d) Local Degree Profile (LDP):* LDP [27] is a simple representation scheme, each node feature summarizes 5 degree statistics information of this node and its 1-hop neighborhood. We selected LDP method as graph centrality measures.

Different centrality measures are applicable in various contexts and provide insights into different aspects of a network's structure and dynamics. Researchers and analysts choose centrality measures based on the specific questions they want to answer and the characteristics of the network under study.

## IV. Proposed Data Augmented GraphCL-Based Static Malware PE Classification

Our proposal is a data augmented GraphCL-based static malware PE classification framework, which can obtain a graph-level representation from malware. We directly extract malware CFG from PE files and through graph contrastive learning obtain a representation of the malware with a vector notation. Finally, malware representations can be performed downstream for various tasks. Graph-level representation shows good performance on malware classification tasks. Next we scrutinize the framework.

### A. Data Augmentation for Graphs

We used the following four data augmentation methods. As shown in Figure 2, our proposal uses two of them as a combination.

*1) Node Dropping:* Randomly discard some parts of the vertex and its connections. The underlying prior enforced by it is that missing part of vertices does not affect the semantic meaning of the graph.The dropping probability of each node follows a default Bernoulli uniform distribution (or any other distribution).

*2) Edge Perturbation:* Randomly add or remove a certain ratio of edges so that the learned representation is consistent under edge perturbation. The prior information of the representation is that adding or removing some edges does not affect the semantics of the graph. The dropping probability of each node follows a default Bernoulli uniform distribution. We only used Edge Removing in this evaluation.

*3) Attribute Masking:* Randomly removing the attribute information of some nodes motivates the model to use other information to reconstruct the masked node attributes. The masking probability of each node feature dimension follows a default uniform distribution. We only used simple Feature Masking.

*4) Subgraph Sampling:* Use random walk subgraph sampling [28] to extract subgraphs from the original graph. The basic assumption is that a graph's semantic information can be preserved in its local structure.

Table I overviews the data augmentation for graphs. The default augmentation (dropping, perturbation, masking) ratio is set to 0.1, and the walk length is set to 10.

TABLE I. OVERVIEW OF DATA AUGMENTATION FOR GRAPHS

| Data Augmentation | Type | Default Setting |
|---|---|---|
| Node Dropping | Nodes, Edges | Bernoulli distribution (ratio = 0.1) |
| Edge Perturbation | Edges | Bernoulli distribution (ratio = 0.1) |
| Attribute Masking | Nodes | Uniform distribution (ratio = 0.1) |
| Subgraph Sampling | Nodes, Edges | Random Walk (length = 10) |

### B. Graph Contrastive Learning

Motivated by recent developments in graph contrastive learning, we propose a graph contrastive learning framework for malware classification. As shown in Figure 2, in graph contrastive learning, pre-training is performed by maximizing the agreement between two augmented views of the same graph by contrastive loss in the potential space. The framework consists of the following four main components:

*1) Graph Data Augmentation:* Throughout the GraphCL framework, given graph data $G$, two related augmented graphs, $\hat{G}_i, \hat{G}_j$, are generated as positive sample pairs by data augmentation.

*2) GIN-based Encoder:* GIN-based encoder $f(\cdot)$ is used to generate graph-level vector representation. There are three layers in the GIN-based encoder, and the hidden layer has 64 dimensions. Through the readout function, the embedding of all the nodes is summed to obtain initial graph representation $h_i, h_j$ for augmented graphs $\hat{G}_i, \hat{G}_j$. Graph contrastive learning does not apply any constraint to the GIN-based encoder.

*3) Projection Head:* Nonlinear transformation $g(\cdot)$, called a projection head, maps the augmented representations to another latent space. Contrast loss is computed in the latent space, and $z_i, z_j$ are obtained by applying a two-layer perceptron (MLP).

*4) Contrastive Loss Function:* Contrastive loss function $\mathcal{L}(\cdot)$ is defined to enforce the maximum consistency between positive pairs $z_i, z_j$ and negative pairs. Here we exploit the normalized temperature-scale cross-entropy loss (NT-Xent) [29] [30] and obtain a graph-level final representation of $z_G$.

## C. Graph Classification

By pre-training with GraphCL, we can obtain a valid graph representation $z_G$. To further verify the effectiveness of our method, different classification models can be chosen for the process, such as random forest, logistic regression, SVM, etc. We chose C-Support Vector Classification (SVC) as the algorithm to validate our pre-trained model's effectiveness.

## V. Implementation Details

We verified the effectiveness of our proposed contrastive learning framework by implementing it with open-source libraries. The implementation details are introduced in this section.

## A. Malware Graph Datasets

The publicly available dataset of malicious software graphs is extremely scarce and does not directly provide the raw binary files. In order to extract CFG, we have constructed our own dataset. Additionally, some publicly available malicious software graph datasets, such as MALNET dataset [31], do not include node features in their graph data themselves, requiring us to generate them based on our needs.

*1) MGD-MULTI Dataset:*

*a) PE Files Source:* Our PE file sample was obtained from the BODMAS Malware Dataset [32]. The software types of all the PE file samples used in our dataset are executable files under an x86-architecture Windows platform without any Dynamic Link Library (DLL) type.

*b) Dataset Description:* From the BODMAS dataset, we selected eight families of malware and took 500 samples from each family, for a total of 4000 samples in our dataset. Our dataset is named MGD-MULTI. The malware family distribution information is shown in Table II.

TABLE II. MALWARE FAMILY DISTRIBUTION OF MGD-MULTI

| Family | Category | Origin Count | Selected Count |
|---|---|---|---|
| sfone | worm | 4729 | 500 |
| upatre | trojan | 3901 | 500 |
| wabot | backdoor | 3673 | 500 |
| benjamin | worm | 1071 | 500 |
| musecador | trojan | 1054 | 500 |
| padodor | backdoor | 655 | 500 |
| gandcrab | ransomware | 617 | 500 |
| dinwod | dropper | 509 | 500 |
| Total | - | 16209 | 4000 |

Due to the difficulty of collecting benign samples and the imbalanced data problem, we did not include benign samples in our multi-class dataset. In our previous malware detection work [33], the MGD-BINARY dataset contained benign samples. We used almost the same GIN model to represent the PE samples, with a slightly different operation of the READOUT layer this time compared to the GIN model in our previous work, giving the final representation a higher vector dimensionality. Based on our previous research, we believe that the GIN model can effectively distinguish benign samples from malicious ones.

Among the different types of malware, we chose families that are more common and have a relatively large number in BODMAS. Due to some limitations of the CFG extraction tool for the PE files we used, many samples couldn't be recognized, causing extraction failure. In addition, for large PE file samples, the process of extracting CFG is very time-consuming. Since the extraction of some samples will fail, we selected a family with more than 500 samples in BODMAS and relatively small original PE files. We further improved the efficiency by only selecting successful samples whose total extraction time is less than 20 seconds in which the total extraction time includes the time of the feature vectors generated by the pre-trained language model.

*c) Dataset Splitting:* We split 4000 pieces of data in MGD-MULTI into training, validation, and testing sets of 50%, 20%, and 30%, respectively. Since the results of the validation set and the test are similar, only the testing set results are shown.

*2) MALNET-TINY Dataset:*

*a) APK Files Source:* The dataset's authentic APK files were acquired from the renowned AndroZoo repository [46, 44]. AndroZoo is a growing collection of Android Applications collected from several sources, including the official Google Play app market.

*b) Dataset Description:* MalNetTiny contains 5,000 malicious and benign software FCGs across 5 different types. Each graph contains at most 5k nodes.

*c) Dataset Splitting:* The dataset is stratified and divided into three sets: training, validation, and test, with a split ratio of 70/10/20, respectively. To thoroughly assess both the model's performance and the dataset's quality, we conduct 10-fold cross-validation on MALNET-TINY, which is distinct from the MGD-MULTI dataset.

## B. Graph Node Feature Generation

For our self-constructed dataset MGD, it is essential to transform the disassembly code within CFG basic blocks into embedded vectors. To achieve this, we opted to utilize a pre-trained language model to generate node embedding vectors. However, for the MALNET-TINY dataset, since the graph data itself does not include node features, we need to extract them directly from the graphs. Therefore, we selected the Local Degree Profile (LDP) method to generate node feature vectors. Next, we will provide a detailed explanation of these two approaches.

*1) Pre-trained Language Model MiniLM:* SentenceTransformers is a python framework for state-of-the-art sentence, text, and image embeddings. The initial work was described in a paper from the Sentence-Bidirectional Encoder Representations from Transformers (Sentence-BERT) [34]. We used the MiniLM model provided by the SentenceTransformers library with the model name, all-MiniLM-L6-v2. The model details used in this paper are shown in Table III.

TABLE III. PRE-TRAINED MINILM MODEL DETAILS

| Name | all-MiniLM-L12-v2 |
|---|---|
| Base Model | microsoft/MiniLM-L12-H384-uncased |
| Max Sequence Length | 256 |
| Dimensions | 384 |
| Normalized Embeddings | true |
| Size | 120 MB |
| Pooling | Mean Pooling |
| Training Data | 1B+ training pairs |

*2) LDP:* PyTorch Geometric (PyG) [35] is a popular Python library built on top of PyTorch, specifically designed for deep learning on graphs and other irregular data structures. PyG provides a wide range of tools and utilities to simplify the implementation of graph neural networks (GNNs) and other graph-based machine learning models. We use the graph transforms to generate the LDP node features.

## C. Graph Contrastive Learning

PyGCL [36] is a PyTorch-based open-source Graph Contrastive Learning library, which features modularized GraphCL components from published papers, standardized evaluation, and experiment management. The Data dataset statistics and hyper-parameters are shown in Table IV.

## VI. Evaluation and Discussion

In this section, we apply the GraphCL model and discuss the experiment results and limitations of our method.

## A. Evaluation Metric

We used the following evaluation metrics to assess the performance of our proposed models:

- **The Micro-averaged F1 score** is defined as the harmonic mean of the precision and recall:

$$Micro\ F1\text{-}score = 2 \times \frac{\text{Micro-Precision} \times \text{Micro-Recall}}{\text{Micro-Precision} + \text{Micro-Recall}}$$

TABLE IV. DATASET STATISTICS AND HYPER-PARAMETERS

| Dataset | MGD-MULTI | MALNET-TINY |
|---|---|---|
| # Graphs | 4000 | 5000 |
| # Avg. Nodes | 3861.7 | 1410.3 |
| # Avg. Edges | 5494.8 | 7125.7 |
| # Features | 384 | 5 |
| # Class | 8 | 5 |
| Learning Rate | 0.0001 | 0.0001 |
| Batch Size | 128 | 256 |
| Epoch | 100 | 100 |
| Hidden Size | 64 | 64 |
| Hidden Layers | 3 | 3 |
| Global Pooling | Sum Pooling | Max Pooling |

- **The Macro-averaged F1 score** is defined as the average of the class-wise/label-wise F1-scores:

$$Macro\ F1\text{-}score = \frac{1}{N}\sum_{i=0}^{N} F1\text{-}score_i$$

where *i* is the class/label index and *N* is the number of classes/labels.

## B. Evaluation Results

Next we apply the GraphCL model and discuss the experiment results of our method. We selected five different data augmentation methods: Identical (*I*), Edge Removing (*ER*), Node Dropping (*ND*), Feature Masking (*FM*), and Random Walk Subgraph (*RWS*).

To compare the different data augmentation approaches on the GraphCL model, we used both data augmentation approaches for the input graph itself *I* + *I* as the GraphCL model baseline. We also tried different combinations of data augmentation, such as *ER* and *ND*, *FM* and *ND*, *FM* and *ER*, *RWS* and *ER*, *RWS* and *ND*, and *RWS* and *FM*. The experimental results on two datasets are shown in Table V.

*1) MGD-MULTI Classification Results:* The best two data augmentation combinations on MGD-MULTI dataset were *RWS* and *FM*. When the walk length of *RWS* is 10 and the ratio of *FM* is 0.1, we obtained the best Micro-F1 (0.9958) and Macro-F1 (0.9959).

In the previous set of experiments, we found that the best data augmentation combination is *RWS* + *FM*. Based on this combination, we also investigated the results on different ratios on the *FM* side, and the *FM* results on different ratios are shown in Table VI. We set the walk length of *RWS* to 10 and adjusted the ratio of the *FM*. When the ratio of *FM* is less than 0.3, the results gradually improve, and as it exceeds 0.3, the results gradually worsen. The optimal result is achieved when the *FM* is set to 0.3.

The *RWS* + *FM* method is most effective because neither method changes the structural information of the original graph. The *RWS* method samples a subgraph that is smaller than the structure of the original graph, but still retains most of the original graph's structure. For the *FM* method, the original graph structure is not changed at all, but the values of some dimensions of the node feature vectors are masked, which

TABLE V. EXPERIMENTAL RESULTS OF DIFFERENT DATA AUGMENTATION

| Augmentation 1 | Augmentation 2 | MGD-MULTI | | MALNET-TINY | |
|---|---|---|---|---|---|
| | | Micro-F1 | Macro-F1 | Micro-F1 | Macro-F1 |
| I | I | 0.9883 | 0.9883 | 0.8758 ± 0.0148 | 0.8753 ± 0.0149 |
| ER[1](0.1) | ND[1](0.1) | 0.9925 | 0.9924 | **0.8652 ± 0.0129** | **0.8641 ± 0.0131** |
| FM[1](0.1) | ND (0.1) | 0.9942 | 0.9942 | 0.8536 ± 0.0150 | 0.8520 ± 0.0156 |
| FM (0.1) | ER (0.1) | 0.9942 | 0.9942 | 0.8394 ± 0.0108 | 0.8384 ± 0.0111 |
| RWS [2] | ER (0.1) | 0.9950 | 0.9949 | 0.7766 ± 0.0152 | 0.7695 ± 0.0155 |
| RWS | ND (0.1) | 0.9950 | 0.9949 | 0.7834 ± 0.0139 | 0.7771 ± 0.0152 |
| RWS | FM (0.1) | **0.9958** | **0.9959** | 0.7760 ± 0.0212 | 0.7715 ± 0.0218 |

[1] Default ratio setting is 0.1.
[2] RWS uses a default walk length setting of 10.

makes the node features more robust. On the contrary, the other two methods (*ER* and *ND*) change the original graph structure more, so the results are lowered.

TABLE VI. BEST COMBINATION WITH DIFFERENT RATIO RESULTS

| Augmentation 1 | Augmentation 2 | MGD-MULTI | |
|---|---|---|---|
| | | Micro-F1 | Macro-F1 |
| RWS [1] | FM (0.1) | 0.9958 | 0.9959 |
| RWS | FM (0.2) | 0.9967 | 0.9967 |
| RWS | FM (0.3) | **0.9975** | **0.9976** |
| RWS | FM (0.4) | 0.9958 | 0.9958 |
| RWS | FM (0.5) | 0.9942 | 0.9941 |

[1] RWS uses a default walk length setting of 10.

*2) MALNET-TINY Classification Results:* The best two data augmentation combinations on MALNET-TINY dataset were $I$ and $I$. On the MGD-MULTI dataset, we did not attempt to explore more combinations with the same settings or different ratios of combinations. To further validate different scenarios, we applied more identical combinations, such as *ND + ND*, *ER + ER*, *RWS + RWS* and *FM + FM* on the new dataset MALNET-TINY. Additionally, concerning different ratios of combinations, we experimented with values ranging from 0.1 to 1.0. For the *RWS*, we tested walk length from 10 to 50,000. The optimal results are shown in Table VII. The best results were achieved when the ratio of all augmentation combinations was set to 0.1. The optimal *RWS* walk length was found to be 20000.

For the same augmentation combinations, the best results were obtained with *I + I*. *ND (0.1) + ND (0.1)* and *ER (0.1) + ER (0.1)* also showed good performance, but *FM (0.1) + FM (0.1)* performed poorly. We believe that the reason behind the poor performance of *FM (0.1) + FM (0.1)* is that the MALNET-TINY dataset has a feature dimension of only 5. Applying the *FM* augmentation can lead to the loss of already limited node feature information, making it difficult for the model to effectively distinguish node features and resulting in poor results. In contrast, *ND* and *ER* operations primarily focus on adjusting the graph structure, which is why their results are more favorable.

It seems that among the different data augmentation combinations, the combination of *ER (0.1) + ND (0.1)* yielded the best performance. This result indicates that using *ER (0.1) + ND (0.1)* in combination provided the most effective data augmentation strategy for improving model performance on the MALNET-TINY dataset. It appears that regardless of using the same data augmentation or different data augmentation strategies, the performance of the *RWS* augmentation is not satisfactory.

This could be due to the specific feature and properties of the MALNET-TINY dataset, which might be better suited for the specific augmentation combination rather than other combinations that involve more complex operations like *RWS*. It's essential to consider the nature of the dataset and the impact of different augmentation techniques on the model's ability to learn meaningful patterns and features from the data. Depending on the dataset's characteristics, some augmentation combinations may be more effective than others in improving model performance. Further experimentation and analysis can help to better understand the relationship between data augmentation and model performance on the specific dataset.

TABLE VII. BEST COMBINATION WITH DIFFERENT RATIO RESULTS

| Augmentation 1 | Augmentation 2 | MALNET-TINY | |
|---|---|---|---|
| | | Micro-F1 | Macro-F1 |
| I | I | 0.8758 ± 0.0148 | 0.8753 ± 0.0149 |
| ND (0.1) | ND (0.1) | **0.8610 ± 0.0173** | **0.8595 ± 0.0178** |
| ER (0.1) | ER (0.1) | 0.8568 ± 0.0110 | 0.8554 ± 0.0110 |
| RWS[1] | RWS | 0.8504 ± 0.0081 | 0.8476 ± 0.0084 |
| FM (0.1) | FM (0.1) | 0.8178 ± 0.0153 | 0.8138 ± 0.0158 |
| I | ER (0.1) | 0.8578 ± 0.0112 | 0.8565 ± 0.0118 |
| I | ND (0.1) | 0.8536 ± 0.0156 | 0.8520 ± 0.0158 |
| I | RWS | 0.8480 ± 0.0116 | 0.8450 ± 0.0121 |
| I | FM (0.1) | 0.8346 ± 0.0118 | 0.8327 ± 0.0127 |
| ER (0.1) | ND (0.1) | **0.8652 ± 0.0129** | **0.8641 ± 0.0131** |
| FM (0.1) | ND (0.1) | 0.8536 ± 0.0150 | 0.8520 ± 0.0156 |
| FM (0.1) | ER (0.1) | 0.8394 ± 0.0108 | 0.8384 ± 0.0111 |
| RWS | ER (0.1) | 0.8466 ± 0.0110 | 0.8441 ± 0.0113 |
| RWS | ND (0.1) | 0.8350 ± 0.0093 | 0.8322 ± 0.0091 |
| RWS | FM (0.1) | 0.8304 ± 0.0155 | 0.8262 ± 0.0161 |

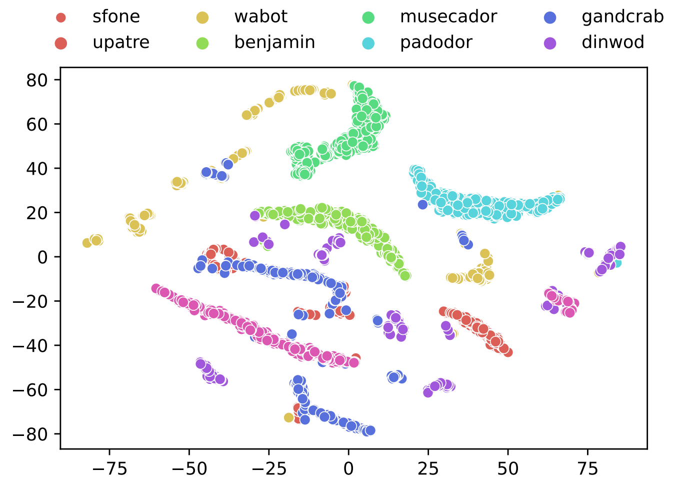
[1] RWS uses the best walk length setting of 2000.

*3) Comparison of Results from Different Datasets:* As shown in Table IV, the graph data of the two datasets have different statistical characteristics. Compared with MGD-MULTI dataset, MALNET-TINY dataset has fewer average nodes and more average edges. The graph samples of the two datasets are quite different. When the node feature vector dimension is

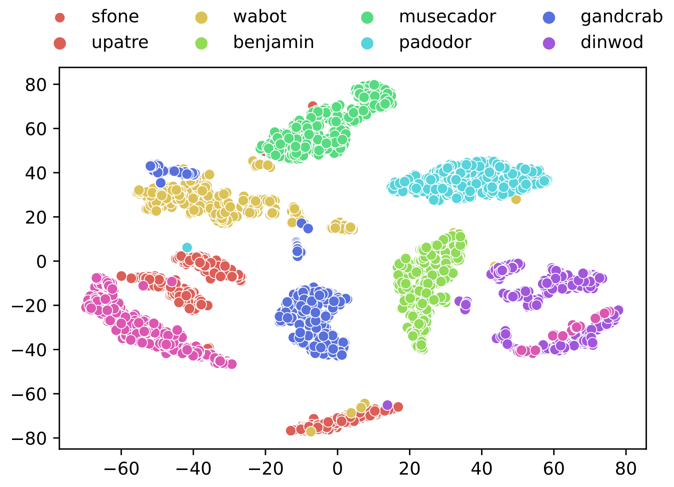
TABLE VIII. EXPERIMENTAL RESULTS OF DIFFERENT LEARNING METHODS

| Name | Method | Type | Classifier | MGD-MULTI | | MALNET-TINY | |
|---|---|---|---|---|---|---|---|
| | | | | Micro-F1 | Macro-F1 | Micro-F1 | Macro-F1 |
| Baseline 1 | GIN-Encoder | - | SVC | 0.9617 | 0.9620 | 0.7060 ± 0.0156 | 0.7012 ± 0.0149 |
| Baseline 2 | GIN (Previous work [33]) | SL[1] | MLP | 0.9958 | 0.9957 | 0.8080 | 0.8126 |
| Proposal | GraphCL | SSL[2] | SVC | 0.9975 | 0.9976 | **0.8758 ± 0.0148** | **0.8753 ± 0.0149** |

[1] SL denotes supervised learning.
[2] SSL denotes self-supervised learning.



(a) Baseline 1 on MGD-MULTI dataset: GIN-Encoder



(b) Proposal on MGD-MULTI dataset: GraphCL (*RWS+FM_0.3*)

Figure 3. t-SNE projection of Baseline 1 and Proposal on MGD-MULTI Dataset

high and the number of edges is relatively small, *FM* becomes more effective. Compared with *RWS*, the more average edges of dataset, the more effective *ER* is.

*4) Comparison of Different Learning Methods:* Our previous studies focused on supervised learning. This study is a graph contrastive learning method in an self-supervised setting. Baseline 1 involves using GIN as the encoder to directly perform graph-level encoding on the input graphs, followed by evaluating the embedding effectiveness using SVC.

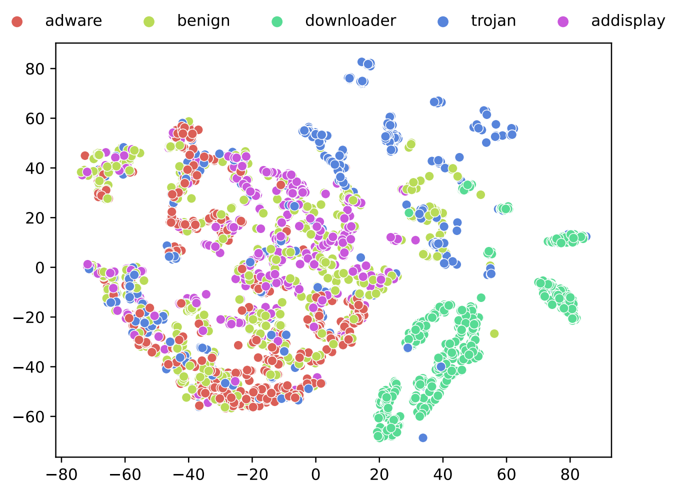
Baseline 2 is our previous work [33] on the malware graph classification task. We utilized the Graph Isomorphism Network for training in a self-supervised setting and directly connected two Multi-Layer Perceptron layers after the GIN readout layer for classification. The GIN model consists of three layers with a hidden layer size of 64, and the MLP has two layers with a hidden layer size of 128. A comparison of different learning type methods is shown in Table VIII. It is worth noting that on the MALNET-TINY dataset, Baseline 2 did not use 10-fold cross-validation, and we only reported the results on the testing set.

On the MGD-MULTI dataset, GraphCL with a setting of *RWS + FM (0.3)* achieved the best classification results. On the MALNET-TINY dataset, GraphCL with a setting of *I + I* achieved the best classification results. The evaluation results indicate that our approach achieved Micro-F1 scores of 0.9975 and Macro-F1 scores of 0.9976 on MGD-MULTI dataset. Similarly, on MALNET-TINY dataset, we obtained Micro-F1 scores of 0.8758 ± 0.0148 and Macro-F1 scores of 0.8753 ± 0.0149. According to our experimental evaluation, the self-supervised learning approach outperformed the supervised learning approach in Graph Neural Networks based on malware classification. We can see that self-supervised learning has great potential.

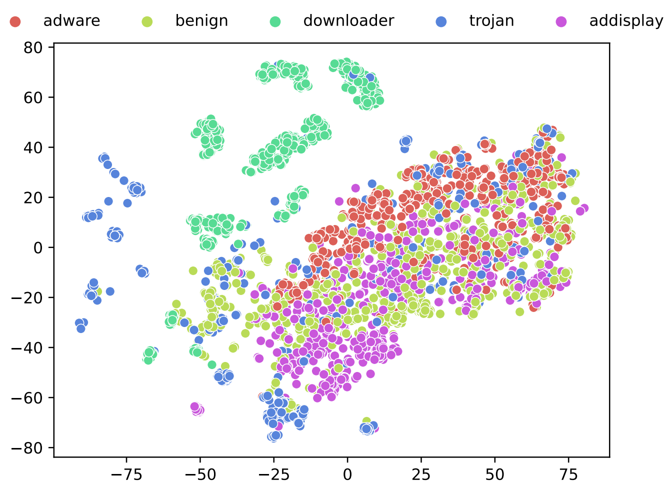
We employed t-SNE technique to visualize the embeddings of Baseline 1 and our proposed method separately on two datasets: MGD-MULTI and MALNET-TINY. For the MGD-MULTI dataset, as shown in Figure 3, the method of Baseline 1 has already clustered some categories, such as the malware of the “padodor” family, but it cannot cluster the “gandcrab” family well. On the other hand, our contrastive learning model proposal can better cluster different categories in the eight classes, and a large distance between different categories is maintained. For the MALNET-TINY dataset, as shown in Figure 4, Compared to the MGD dataset, the visualization results of the TINY dataset are not satisfactory. We found that among the four classes of malicious software, the performance for the “trojan” type is the worst, while the “downloader” type has the best results. For “benign” samples, it is challenging to distinguish them from “adware” and “addisplay”.

## *C. Current Limitations*

In this study, the two datasets used were relatively small. We achieved promising results on the MGD-MULTI dataset. However, the performance on the MALNET-TINY dataset was not very satisfactory.



(a) Baseline 1 on MALNET-TINY dataset: GIN-Encoder



(b) Proposal on MALNET-TINY dataset: GraphCL ($I$ + $I$)

Figure 4. t-SNE projection of Baseline 1 and Proposal on MALNET-TINY Dataset

*1) MGD-MULTI dataset:* In the feature engineering stage, there are two steps that are very time-consuming. Firstly, extracting the CFG of the PE file is very time-consuming. Secondly, during the construction of training data, it is also very time-consuming because it requires embedding representation of all nodes' features in the graphs using a pre-trained model. During the training phase, due to the large graph data structures and the high dimensionality of each node in the graph (384 dimensions), the GraphCL model training is slow, even though the dataset itself contains only 4000 samples. We desire a better way to generate node features, such as a lower dimensional in a method that retains its effectiveness.

*2) MALNET-TINY dataset:* GraphCL ($I$ + $I$) is a combination of two Identical, and the effect is equivalent to turning a training set of N samples into 2N samples. The same data model is learned twice for the same data, so the obtained result naturally outperforms GIN-Encoder. Compared to other data augmentation combinations, the combination of $I$ + $I$ yields the best results, and a reasonable explanation for this has not been found yet.

*3) Graph Self-supervised Learning:* GraphCL is representative of contrastive methods, but it overly relies on high-quality data augmentation. We explored the effects of various data augmentation methods, such as *FM*, *RWS*, and *ER*. We found that effective data augmentation on graphs often depends on domain knowledge. For instance, *ER* is beneficial for training on the MALNET-TINY dataset, but it has a negative impact on the MGD-MULTI dataset. However, in the context of graph contrastive learning, there is still no universally effective data augmentation method up to now. Generative self-supervised learning can avoid the aforementioned dependencies, as it aims to reconstruct the features and information of the data itself, such as GraphMAE [21] and WGDN [22] models.

## VII. CONCLUSION

We propose using graph contrastive learning for unsupervised learning of different families of malicious software. We employ SVC for multi-class classification of the graph representations and achieve promising results. For our self-built dataset MGD-MULTI, we extract the CFG of malicious software and embed the disassembly code in CFG basic blocks using a pre-trained large-scale language model, MiniLM. For the publicly available MALNET-TINY dataset, as it does not provide node features for the graph data based on FCG, we generate node representations using LDP. Through these two methods, both CFG-based and FCG-based malicious software are transformed into directed graphs with node features. Using graphs to represent malicious software offers significant advantages, as each node carries rich information and preserves the structural details of the samples. Graph models can learn from both global and local perspectives. Furthermore, self-supervised learning eliminates the need for sample annotations, learning from self-supervised signals, making it more efficient to leverage a larger number of samples for pre-training. On our self-built dataset, the graph-based self-supervised method for malicious software classification has outperformed supervised graph learning methods, such as Graph Isomorphism Networks used for graph classification. In future work, our focus will shift towards self-supervised learning.

## REFERENCES

[1] Y. Gao, H. Hasegawa, Y. Yamaguchi, and H. Shimada, “Unsupervised graph contrastive learning with data augmentation for malware classification,” in *Proceedings of the 16th International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2022*, pp. 41–47, IARIA, 2022.

[2] H. S. Anderson and P. Roth, “EMBER: an open dataset for training static PE malware machine learning models,” *CoRR*, vol. abs/1804.04637, pp. 1–8, 2018.

[3] B. Athiwaratkun and J. W. Stokes, “Malware classification with LSTM and GRU language models and a character-level CNN,” *ICASSP 2017*, pp. 2482–2486, 2017.

[4] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, “Large-scale malware classification using random projections and neural networks,” *ICASSP 2013*, pp. 3422–3426, 2013.

[5] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, “Malware classification with recurrent networks,” *ICASSP 2015*, pp. 1916–1920, 2015.

[6] F. Cohen, “Computer Viruses: Theory and Experiments,” *Computers & security, Vol. 6, No. 1*, pp. 22–35, 1987.

[7] J. O. Kephart, G. B. Sorkin, W. C. Arnold, D. M. Chess, G. Tesauro, and S. R. White, “Biologically Inspired Defenses Against Computer Viruses,” *IJCAI 1995, Vol. 2*, pp. 985–996, 1995.

[8] J. Saxe and K. Berlin, “Deep neural network based malware detection using two dimensional binary program features,” *MALWARE 2015*, pp. 11–20, 2015.

[9] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, “Malware Detection by Eating a Whole EXE,” *AAAI Workshops 2018*, pp. 268–276, 2018.

[10] E. Raff, J. Sylvester, and C. Nicholas, “Learning the PE Header, Malware Detection with Minimal Domain Knowledge,” *AISec@CCS 2017*, pp. 121–132, 2017.

[11] Y. Chen, S. Wang, D. She, and S. Jana, “On Training Robust PDF Malware Classifiers,” *USENIX Security 2020*, pp. 2343–2360, 2020.

[12] S. E. Coull and C. Gardner, “Activation Analysis of a Byte-Based Deep Neural Network for Malware Classification,” *SP Workshops 2019*, pp. 21–27, 2019.

[13] E. M. Rudd, F. N. Ducau, C. Wild, K. Berlin, and R. E. Harang, “ALOHA: Auxiliary Loss Optimization for Hypothesis Augmentation,” *USENIX Security 2019*, pp. 303–320, 2019.

[14] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, “CADE: Detecting and Explaining Concept Drift Samples for Security Applications,” *USENIX Security 2021*, pp. 2327–2344, 2021.

[15] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic Graph CNN for Learning on Point Clouds,” *ACM Transactions on Graphics (TOG), Vol. 38, Issue 5*, pp. 1–12, 2019.

[16] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?,” in *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)*, pp. 1–17, 2019.

[17] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, “graph2vec: Learning Distributed Representations of Graphs,” *CoRR*, vol. abs/1707.05005, pp. 1–8, 2017.

[18] F. Sun, J. Hoffmann, V. Verma, and J. Tang, “Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization,” in *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*, pp. 1–16, 2020.

[19] M. Dib, S. Torabi, E. Bou-Harb, N. Bouguila, and C. Assi, “Evoliot: A self-supervised contrastive learning framework for detecting and characterizing evolving IoT malware variants,” in *Proceedings of ASIA CCS ’22: ACM Asia Conference on Computer and Communications Security*, pp. 452–466, 2022.

[20] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, “Graph contrastive learning with augmentations,” in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

[21] Z. Hou, X. Liu, Y. Cen, Y. Dong, H. Yang, C. Wang, and J. Tang, “Graphmae: Self-supervised masked graph autoencoders,” in *Proceedings of KDD ’22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 594–604, 2022.

[22] J. Cheng, M. Li, J. Li, and F. Tsung, “Wiener graph deconvolutional network improves graph self-supervised learning,” in *Proceedings of the 37th AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023*, pp. 7131–7139, AAAI Press, 2023.

[23] I. You and K. Yim, “Malware obfuscation techniques: A brief survey,” in *Proceedings of the 15th International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA*, pp. 297–300, IEEE Computer Society, 2010.

[24] J. Jang, S. Choi, and J. Hong, “A method for resilient graph-based comparison of executable objects,” in *Research in Applied Computation Symposium, RACS ’12*, pp. 288–289, ACM, 2012.

[25] V. S. Sathyanarayan, P. Kohli, and B. Bruhadeshwar, “Signature generation and detection of malware families,” in *Proceedings of 13th Australasian Conference on Information Security and Privacy, ACISP*, vol. 5107 of *Lecture Notes in Computer Science*, pp. 336–349, Springer, 2008.

[26] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, “MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers,” *CoRR*, vol. abs/2002.10957, pp. 1–15, 2020.

[27] C. Cai and Y. Wang, “A simple yet effective baseline for non-attribute graph classification,” *CoRR*, vol. abs/1811.03508, pp. 1–13, 2018.

[28] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864, 2016.

[29] A. van den Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding,” *CoRR*, vol. abs/1807.03748, pp. 1–13, 2018.

[30] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, “Graph contrastive learning with augmentations,” *CoRR*, vol. abs/2010.13902, pp. 1–12, 2020.

[31] S. Freitas, Y. Dong, J. Neil, and D. H. Chau, “A large-scale database for graph representation learning,” in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021*, 2021.

[32] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, “BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware,” *DLS 2021*, pp. 78–84, 2021.

[33] Y. Gao, H. Hasegawa, Y. Yamaguchi, and H. Shimada, “Malware detection using attributed cfg generated by pre-trained language model with graph isomorphism network,” in *Proceedings of the 12th IEEE International Workshop on Network Technologies for Security, Administration and Protection (NETSAP 2022)*, pp. 1495–1501, 2022.

[34] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” *CoRR*, vol. abs/1908.10084, pp. 1–11, 2019.

[35] M. Fey and J. E. Lenssen, “Fast graph representation learning with PyTorch Geometric,” in *Proceedings of the ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[36] Y. Zhu, Y. Xu, Q. Liu, and S. Wu, “An empirical study of graph contrastive learning,” *CoRR*, vol. abs/2109.01116, pp. 1–25, 2021.