A2-CLM : Few-shot Malware Detection Based on Adversarial Heterogeneous Graph Augmentation

Anonymous Author(s)

ABSTRACT

Malware attacks, especially "few-shot" malware, have profoundly harmed the cyber ecosystem. Recently, malware detection models based on graph neural networks have achieved remarkable success. However, these efforts over-rely on sufficient labeled data for model training and thus may be brittle in few-shot malware detection because of the label scarcity. To this end, we propose a self-supervised malware detection framework based on graph contrastive learning and adversarial augmentation, termed A2-CLM, to address the challenge of few-shot malware detection. Particularly, A2-CLM first depicts the malware execution context with a sensitivity heterogeneous graph by assessing the security semantic of each behavior. Afterwards, A2-CLM designs multiple adversarial attacks to generate more practical contrastive pairs, including the PGD attack, attribute masking attack, meta-graph-guide sampling attack, direct system calls attack, and obfuscation attack, which is beneficial to strengthening the model's effectiveness and robustness. To alleviate the training workload of contrastive learning, we introduce a momentum strategy to train the multiple graph encoders in A2-CLM. Especially on 1-shot detection tasks, A2-CLM achieves performance gains of up to 24.63% and 4.58% against supervised and self-supervised detection methods, respectively.

CCS CONCEPTS

• **Security and privacy → Few-shot malware detection**.

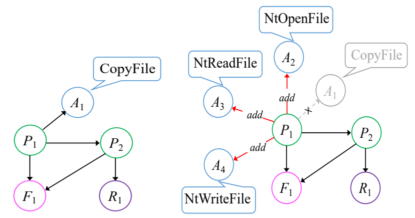
KEYWORDS

few-shot malware detection, security semantic, graph contrastive learning, adversarial heterogeneous graph augmentation

ACM Reference Format:
Anonymous Author(s). 2018. A2-CLM : Few-shot Malware Detection Based on Adversarial Heterogeneous Graph Augmentation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Driven by the advent of sophisticated attack vectors, the exponential increase of new malware seriously disturbs the health of the network environment and degrades the user experience [11, 69]. According to a recent report [4], AV-TEST identified over 450,000



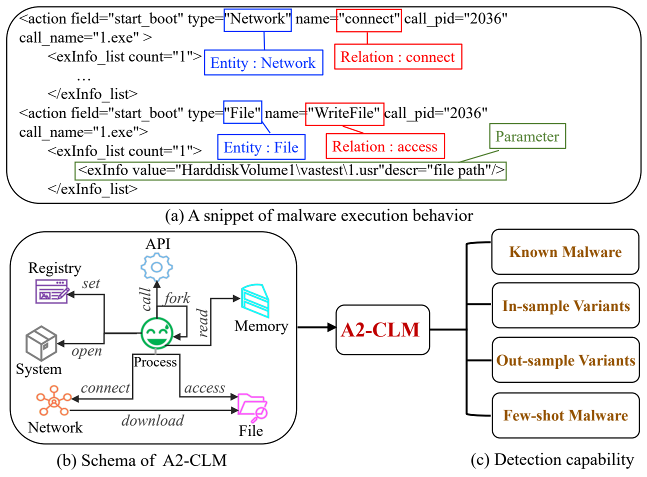
(a) Normal instance (b) Adversarial instance

Figure 1: (a) A normal malware instance; (b) An adversarial malware instance using code obfuscation techniques.

new malware registrations daily, many of which are wild instances of malware or new families that have not yet been seen, making it difficult to collect and label analyzable samples [1]. As a result, effectively detecting malware, particularly newly emerging malware, is critical in network security to protect users from future threats.

Deep learning exhibits a crucial role in malware detection, as it can automatically learn the feature vectors from the malware samples [9, 15]. Generally, these approaches can be roughly divided into two scopes: feature-based and heterogeneous graph-based malware detection methods. Specifically, the feature-based detection methods concentrate on extracting representative signature or behavior features, such as opcodes [17, 40, 64], permissions[3, 8], API call sequences [11, 22, 38, 46, 66], and network traffic [35, 65]. However, these methods merely emphasize the isolated features of the malware and ignore the contextual structural information of malware propagation. Thus, several studies have strove to capture the interactive structure patterns for malware detection by leveraging heterogeneous graphs [14, 29, 30, 49, 55]. These methods model the various malware entities as a heterogeneous graph and employ graph neural networks (GNNs) to learn a more comprehensive low-dimensional representation.

Despite the aforementioned deep learning-based detection methods recently showing the great potential in malware detection, unfortunately, they are mostly plagued by two flaws. Firstly, the existing deep learning-based detection methods severely rely on training a proper model in a supervised end-to-end manner, where a large number of task-specific labels are needed [48, 61]. However, the few-shot issue of malware detection tasks is significant, where each class of the training set contains a limited number of samples [45], which may disable the existing deep learning-based methods. Secondly, the existing deep learning-based detection methods that emphasize grasping attack details of known training samples often lead to poor generalization capabilities and a lack of robustness [13, 67] against adversarial samples shown in Figure 1(b). Actually, an experienced attacker always replaces sophisticated malicious behavior (i.e., the malicious API operation "CopyFile" in Figure 1(a)) with equivalent normal behavior (i.e., the



```
<action field="start_boot" type="Network" name="connect" call_pid="2036"
call_name="1.exe" >
    <exInfo_list count="1">
    ...
    </exInfo_list>
<action field="start_boot" type="File" name="WriteFile" call_pid="2036"
call_name="1.exe">
    <exInfo_list count="1">
      <exInfo value="HarddiskVolume1\vastest\1.usr"descr="file path"/>
    </exInfo_list>
```

Figure 2: Network schema and detection capability of A2-CLM.

benign API operations ("*NtOpenFile*", "*NtReadFile*", "*NtWriteFile*") in Figure 1(b)) to evade detection, resulting in the existing detection models being feeble for defending these advanced variant attacks.

To this end, malware detection models need to become more robust in the face of unseen, even unlabeled, few-shot malware, which leads to our innovations below.

In this paper, we present a self-supervised graph Contrastive Learning few-shot Malware detection framework with Adversarial heterogeneous graph Augmentation (i.e., A2-CLM) to achieve a more robust and effective few-shot malware detection. Concretely, A2-CLM first advocates a sensitivity heterogeneous graph to model malware's interactive behavior (Figure 2(a)) as the contrastive instance (Figure 2(b)), whose key insight is that the pre-determined sensitivity of run-time behavior can assist in accurate few-shot malware detection. Then, to manufacture more practical augmented instances for few-shot malware detection, A2-CLM explores comprehensive adversarial augmentations such as attribute-level adversarial attacks and structure-level adversarial attacks to simulate the case of Figure 1(b), which provides input data with rich and reasonable noise for the subsequent instance-based discriminator. Finally, inspired by the soaring performance of contrastive learning in computer vision [10, 20, 24] and natural language processing [12, 27, 60], A2-CLM introduces self-supervised graph contrastive learning to train the graph encoders to generate robust and powerful representations, which eventually achieve the ability of malware detection with a small number of unlabeled samples (Figure 2(c)).

To conclude, the major contribution of this work can be summarized as follows:

- We present a novel few-shot malware detection framework, termed A2-CLM, to utilize adversarial heterogeneous graph augmentation to contribute to self-supervised graph contrastive learning, which is capable of achieving more effective and robust malware detection since A2-CLM prevents some cases, such as code obfuscation variants, from being misclassified by the detection model.

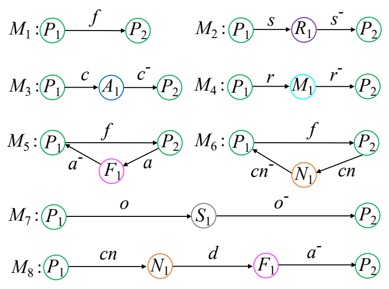

M_1: P_1 f P_2
M_2: P_1 s R_1 s^- P_2
M_3: P_1 c A_1 c^- P_2
M_4: P_1 r M_1 r^- P_2
M_5: P_1 f P_2 a^- F_1 a
M_6: P_1 f P_2 cn^- N_1 cn
M_7: P_1 o S_1 o^- P_2
M_8: P_1 cn N_1 d F_1 a^- P_2

Figure 3: Meta-graphs of A2-CLM.

- We present a sensitivity heterogeneous graph to model malware instances that fully exploits the propagative structure information as well as the security semantics of various malicious interactive behaviors. A sensitivity grading method that integrates statistics-based and clustering-based techniques, in particular, assigns varying degrees of sensitivities to each interactive behavior, which can aid in accurate few-shot malware detection and improve the interpretability of detection results.
- We design two-level adversarial attacks to generate more practical contrastive pairs for few-shot malware, including the PGD attack, attribute masking attack, meta-graph-guide sampling attack, direct system calls attack, and obfuscation attack, each of which imposes certain semantic or structural priors and is beneficial to learning more robust representations.
- Finally, we extensively evaluate A2-CLM on diverse real-world datasets. A2-CLM achieves significant performance gains in accuracy and F1-score compared to state-of-the-art baselines, especially in few-shot malware detection tasks, where it can achieve at least 4.58% and 3.52% improvements on the 1-shot task and 10-shot task, respectively.

2 PRELIMINARIES

Definition 1. ***Few-shot Learning (FSL) [57]*** *Few-shot learning is a type of machine learning (specified by E, T, and P), where E contains only a limited number of examples with supervised information for the target T, usually less than 20.*

Definition 2. ***Sensitivity Heterogeneous Graph of Few-shot Malware (SHGFM).*** *A sensitivity heterogeneous graph* $G = (\mathcal{V}, \mathcal{E}, \mathrm{S})$ *of the few-shot malware with a node type mapping* $\Psi : \mathcal{V} \mapsto \mathcal{T}$ *and an edge type mapping* $\psi : \mathcal{E} \mapsto \mathcal{R}$. *Let* $\mathcal{V}$ *be the set of nodes,* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ *be the set of relationships between nodes in* $\mathcal{V}$, *and* $\mathrm{S} \in \mathbb{R}^{d \times f}$ *is the sensitivity attribute matrix. Each node* $v_x \in \mathcal{V}$ *belongs to one particular malware entity type in the node type set* $\mathcal{T} : \Psi(v_x) \in \mathcal{T}$, *and each edge* $e_x \in \mathcal{E}$ *belongs to a particular relationship type in the edge type set* $\mathcal{R} : \psi(e_x) \in \mathcal{R}$, *where* $|\mathcal{T}| + |\mathcal{R}| > 2$. *The* x*-th row vector* $s_x \in \mathbb{R}^f$ *of the sensitivity attribute matrix denotes the entity attribute feature that concatenates the sensitivity score of node* v_x, *where each sensitivity score belongs to {1, 2, 3} assesses the degree of malice of the corresponding run-time behavior.*

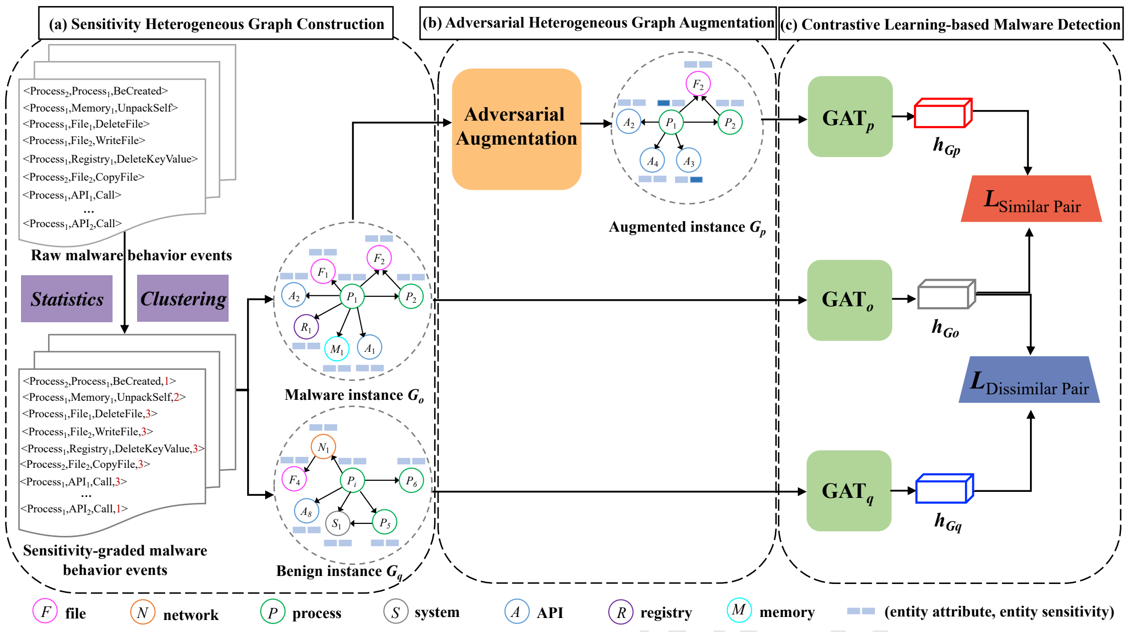

(a) Sensitivity Heterogeneous Graph Construction
(b) Adversarial Heterogeneous Graph Augmentation
(c) Contrastive Learning-based Malware Detection
<Process2,Process1,BeCreated>
<Process1,Memory1,UnpackSelf>
<Process1,File1,DeleteFile>
<Process1,File2,WriteFile>
<Process1,Registry1,DeleteKeyValue>
<Process2,File2,CopyFile>
<Process1,API1,Call>
...
<Process1,API2,Call>
Raw malware behavior events
Statistics
Clustering
<Process2,Process1,BeCreated,1>
<Process1,Memory1,UnpackSelf,2>
<Process1,File1,DeleteFile,3>
<Process1,File2,WriteFile,3>
<Process1,Registry1,DeleteKeyValue,3>
<Process2,File2,CopyFile,3>
<Process1,API1,Call,3>
...
<Process1,API2,Call,1>
Sensitivity-graded malware behavior events
Malware instance G_o
Benign instance G_q
Adversarial Augmentation
Augmented instance G_p
GAT$_p$
GAT$_o$
GAT$_q$
h_{Gp}
h_{Go}
h_{Gq}
$L_{\text{Similar Pair}}$
$L_{\text{Dissimilar Pair}}$
F file
N network
P process
S system
A API
R registry
M memory
(entity attribute, entity sensitivity)

Figure 4: Overview of A2-CLM. (a) Sensitivity Heterogeneous Graph Construction intends to model the sensitivity-graded behavior of the target malware with a sensitivity heterogeneous graph G_o, which holds processes, APIs, files, networks, systems, registries, memories, and their interactive relationships. (b) Adversarial Heterogeneous Graph Augmentation crafts more challenging augmented instances G_p through two levels of adversarial augmentation, such as attribute-level adversarial attacks and structure-level adversarial attacks. (c) Contrastive Learning-based Malware Detection authorizes multiple GAT encoders, which can concurrently obtain the graph-level representations h_{G_o}, h_{G_p}, and h_{G_q}; then, the InfoNCE loss function is encouraged to evaluate the correspondence between the original malware instance and its augmented instances for the ultimate detection.

Definition 3. ***Meta-graph [68].*** *A meta-graph M is a directed acyclic graph with a single source node n_s (i.e., with in-degree 0) and a single target node n_t (i.e., with out-degree 0), defined on a SHGFM $G = (\mathcal{V}, \mathcal{E}, \mathrm{S})$ with schema $T_G = (\mathcal{A}, \mathcal{R})$, then a meta-graph can be defined as $M = (\mathcal{V}_M, \mathcal{E}_M, \mathcal{A}_M, \mathcal{R}_M, n_s, n_t)$, where $\mathcal{V}_M \in \mathcal{V}$, $\mathcal{E}_M \in \mathcal{E}$ are constrained by $\mathcal{A}_M \in \mathcal{A}$ and $\mathcal{R}_M \in \mathcal{R}$, respectively.*

Figure 3 shows eight types of meta-graphs, and different meta-graphs express different semantic information.

3 METHODOLOGY

In this section, we first formalize the few-shot malware detection problem and then elucidate the details of A2-CLM (shown in Figure 4), which includes three components: (1) sensitivity heterogeneous graph construction (Figure 4(a)); (2) adversarial heterogeneous graph augmentation (Figure 4(b)); and (3) contrastive learning-based malware detection (Figure 4(c)).

3.1 Problem Statement

Few-shot Malware Detection. As the means of attack by hackers become more and more sophisticated [1, 54], many new emerging malware attacks make it difficult to collect sufficient analysis samples in the wild, resulting in the "few-shot problem" that is critical in malware detection tasks. Additionally, the security semantics implied by different run-time behaviors are valuable for few-shot malware detection, which are missed by existing studies. For example, in Figure 4(a), the behavior "Process 2036 created by Process 208 starts execution (i.e. ($Process_1$, $Process_2$, BeCreated))" is normal, while the behavior "Process 2036 modifies $HarddiskVolume1\backslash vastest\backslash 1.usr$ (i.e. ($Process_1$, $File_2$, WriteFile))" is malicious. The observations above motivate us to use self-supervised learning and malware fine-grained execution context to improve few-shot malware detection.

Recently, contrastive learning has shown sweeping successes in few-shot learning [56], which concentrates on leveraging the data's inherent co-occurrence relationships as self-supervision without the task-specific labeled information. In this work, we develop a graph contrastive learning few-shot malware detection framework that uses adversarial augmentation to improve the model's robustness with adversarial perturbations in a self-supervised manner.

A2-CLM. Given the target malware's executive behavior events $D_o = \{e_1, \cdots, e_{|D|}\}$, A2-CLM first grades each run-time behavior with varying degrees of sensitivities calculated from the parameter information involved in each event by statistics-based and clustering-based techniques. Then A2-CLM leverages a sensitivity heterogeneous graph G_o to model the sensitivity-graded behavior events D'_o. To craft more challenging contrastive pairs, A2-CLM fully augments m positive instances $\mathbf{G}^o_P = (G^o_{p.1}, \cdots, G^o_{p.m})$ with two levels of adversarial attacks on the original G_o and randomly chooses n negative instances $\mathbf{G}^o_Q = (G^o_{q.1}, \cdots, G^o_{q.n})$ from the rest of the software types (e.g., benign, Trojan.Kazy, and so on). Afterwards, using encoders GAT_o, GAT_p, and GAT_q, A2-CLM learns the graph-level representations of G_o, $\mathbf{G}_P^o$, and $\mathbf{G}_Q^o$. Finally, A2-CLM utilizes the instance discriminators based on contrastive learning to evaluate the agreement of each instance pair and outputs the predicted malware type of the target few-shot malware G_o.

Generally, the critical issues that A2-CML hankers for settling are as follows:

***Issue*$_1$:** How to generate the fine-grained and robust instances for few-shot malware detection tasks?

***Issue*$_2$:** How to create more proper and practical positive and negative contrastive instance for malware heterogeneous graphs?

***Issue*$_3$:** What are the discrimination rules of few-shot malware detection?

3.2 Sensitivity Heterogeneous Graph Construction

As shown in Figure 4(a), we collect the underlying executive behavior of malware in the KingKong system [44], which holds abundant interactive relationships among heterogeneous malware objects (e.g., APIs, processes, networks, etc.). Hence, to address ***Issue*$_1$**, it is insightful to take advantage of the heterogeneous graph to model various heterogeneous malware entities and relationships for few-shot malware detection tasks. Unfortunately, the existing heterogeneous graph-based detection methods, such as MatchGNet [55], MG-DVD [29], and so on, merely emphasize malware object type and interactions among them; they ignore the security semantics implicit in run-time behavioral parameters, which assist in identifying malicious patterns and improve the interpretability of detection results. To this end, A2-CLM is responsible for associating each run-time behavior with crucial degrees of sensitivity by exploiting the security semantics of specific parameters. Concretely, we propose a sensitivity grading method that integrates statistics-based and clustering-based techniques to divide the entire behavior space into three categories, such as benign (i.e., sensitivity of 1), sensitive (i.e., sensitivity of 2), and malicious (i.e., sensitivity of 3), which represent distinct security semantics.

3.2.1 Sensitivity Grading. We first employ a statistics-based method to tackle the run-time behavior by estimating the distribution of behavioral parameters in malicious software and benign software. Concretely, we implement term frequency inverse document frequency (TF-IDF) [19, 58] to assess the sensitivity of each behavior event. Formally, the frequency can be expressed as:

$$TF_i = \frac{n_i}{\sum_k n_k}, \quad (1)$$

where n_i is the number of occurrences of parameter i in events set $\boldsymbol{D}$, and $\sum_k n_k$ is the sum of the occurrences of all parameters in the events set $\boldsymbol{D}$.

Let TF_m denote the frequency of a parameter in malicious software, while DF_m refers to the frequency of the software containing the parameter. TD_m equals $TF_m \times DF_m$. Similarly, TF_b, DF_b, and TD_b denote the corresponding values in benign software. Thereby, for a parameter, if it has a higher TD_m but lower TD_b, it may imply malicious behavior with a high probability, while a parameter with a lower TD_m and a higher TD_b is more likely to be benign.

Table 1: Example of Several Behavior Events And Assigned Sensitivities in Virus:Win32/Shodi.I

Behavior Event	Sensitivity
$\langle Process_{2036}, Process_{208}, BeCreated\rangle$	1
$\langle Process_{2036}, Memory, LoadLibrary\rangle$	1
$\langle Process_{2036}, File, ReadFile\rangle$	2
$\langle Process_{2036}, System, CreateMutex\rangle$	2
$\langle Process_{2036}, File, WritePEFile\rangle$	3
$\langle Process_{1928}, Network, QueryDNS\rangle$	3

For the behavior events uncovered by the statistics-based technique, we utilize a clustering-based technique to further evaluate their sensitivity. Specifically, we make the involved parameters of each behavior event compose a document (d), in which each parameter is a string. Therefore, we can formalize the parameter clustering as short text clustering and utilize the powerful GSDMM [62] to achieve the purpose. Finally, with the statistics-based and clustering-based techniques, the entire malware behavior events space is covered, and they can be divided into K (i.e., $K = 3$) categories following their run-time parameters, which completely correspond to the benign, sensitive, and malicious three types in Table 1.

3.2.2 Sensitivity Heterogeneous Graph Construction. Given the behavior events of the target malware after sensitivity grading, A2-CLM extracts 7 types of malware objects (i.e., process, API, file, system, registry, memory, and network), 8 types of interactive relationships (i.e., $Process \xrightarrow{fork} Process$, $Process \xrightarrow{call} API$, $Process \xrightarrow{access} File$, $Process \xrightarrow{open} System$, $Process \xrightarrow{connect} Network$, $Process \xrightarrow{read} Memory$, $Process \xrightarrow{set} Registry$, and $Network \xrightarrow{download} File$), and 3 types of sensitivity (i.e., benign, sensitive, and malicious) from them, which can roundly characterize the attack patterns of few-shot malware. Then, starting with the target process node Tar, we insert the event e_i into SHGFM, where $e_i.Nei \in V$. Eventually, we obtain the fine-grained sensitivity heterogeneous graph G_o of the target malware, which represents as the adjacency matrix $\boldsymbol{A_o}$ and sensitivity attribute matrix $\boldsymbol{S_o}$.

3.3 Adversarial Heterogeneous Graph Augmentation

Clearly, the performance of contrastive learning is heavily dependent on the design of positive and negative instance pairs, and improper choice of data augmentation can degrade downstream performance [23, 59]. Unfortunately, prior graph contrastive learning efforts highlight the generation of trivial augmented instances for homogeneous graph-structured data by randomly adding or deleting nodes or edges [39, 59], which are not applicable to malware heterogeneous graph instances due to neglecting the nonlinear dependencies of various heterogeneous entities. Hence, to puzzle out ***Issue*$_2$**, we prefer to implement adversarial heterogeneous graph augmentation to compensate for the limitation of the existing graph contrastive learning by generating more challenging positive pairs and effective negative pairs, which is beneficial to improving the model's robustness by investigating the adversarial attack derived

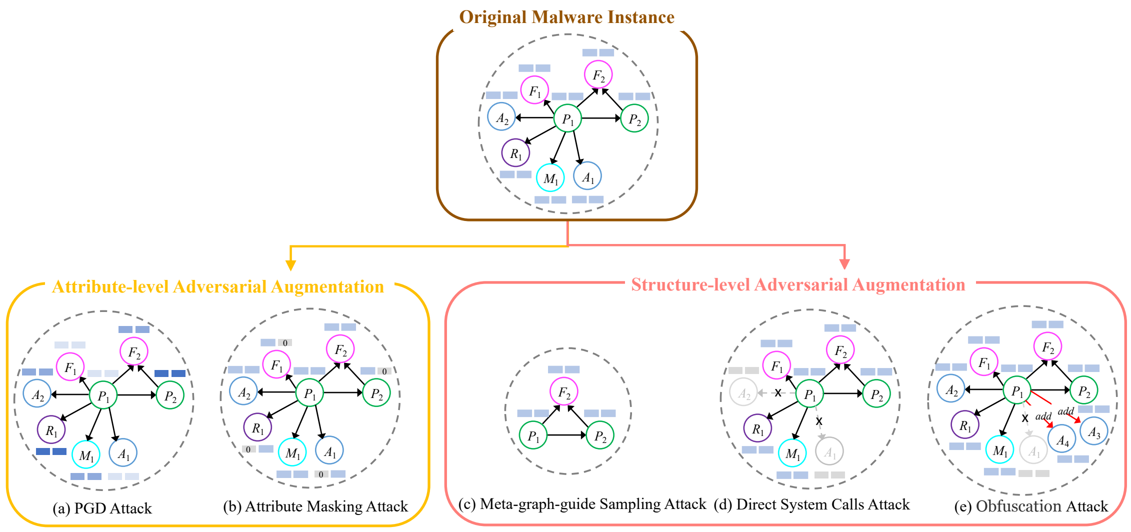

Original Malware Instance
Attribute-level Adversarial Augmentation
Structure-level Adversarial Augmentation
(a) PGD Attack
(b) Attribute Masking Attack
(c) Meta-graph-guide Sampling Attack
(d) Direct System Calls Attack
(e) Obfuscation Attack
add
add

Figure 5: Five types of adversarial heterogeneous graph augmentations. (a) PGD Attack; (b) Attribute Masking Attack; (c) Meta-graph-guide Sampling Attack; (d) Direct System Calls Attack; (e) Obfuscation Attack.

from malware domain knowledge. Concretely, as shown in Figure 5, we design five malware adversarial heterogeneous graph augmentations by modifying graph structure (i.e., structure-level adversarial augmentation) and node attributes (i.e., attribute-level adversarial augmentation), including the PGD attack, attribute masking attack, meta-graph-guide sampling attack, direct system calls attack, and obfuscation attack, which can be utilized to train a robust graph contrastive learning model for few-shot malware detection.

3.3.1 PGD Attack. To perform the transformations on the sensitivity attribute matrix $\boldsymbol{S}$, we first project the attributed features of all nodes in SHGFM to a common space since nodes in SHGFM have different types. Formally, for each node $v_x \in \mathcal{V}$ with type $T_x \in \mathcal{T}$ and sensitivity attribute feature $s_x \in \mathrm{S}$, the projection function $g(\cdot)$ is defined as follows:

$$s'_x \;=\; g(s_x) \;=\; s_x W_{T_x}, \tag{2}$$

where s'_x is the transformed node sensitivity attribute feature, and W_{T_x} is the projection weight matrix for node type T_x ($T = 7$).

As shown in Figure 5(a), with the processed s'_x, we apply the project gradient descent (PGD) attack [32] to infuse perturbation δ into the node sensitivity attribute features of SHGFM. Concretely, the attribute feature attacked by PGD can be denoted as

$$\hat{s}'_x \;=\; s'_x + \delta, \tag{3}$$

where the perturbation δ can be optimized as follows:

$$\delta = arg\ max_{\|\delta'\|_{n \le \epsilon}}\ \mathcal{L}\ (\theta, s'_x + \delta'), \tag{4}$$

where $\|\cdot\|_n$ denotes the l_n-norm distance metric, ϵ is the perturbation budget, θ is the model parameters, and $\mathcal{L}$ is the contrastive loss function. Finally, we obtain the first kind of augmented instance $G^o_{p.1}$=($\mathcal{V}$, $\mathcal{E}$, S + δ) of the original few-shot malware instance G_o.

3.3.2 Attribute Masking Attack. The attribute masking attack is another attribute-level adversarial augmentation, as shown in Figure 5(b). Encouraged by the significant effect of node attribute masking on homogeneous graphs [63], we extend it to heterogeneous graphs to generate more elegant positive instances for few-shot malware by performing the transformation on the processed sensitivity attribute matrix $\mathrm{S_o}$. Formally, we have

$$\mathcal{H}^{mask}(\mathrm{S'}_o) = \mathrm{S}_o * (1 - \mathrm{L}_m) + \mathrm{V} * \mathrm{L}_m, \tag{5}$$

where $*$ is the element-wise multiplication; $\boldsymbol{L_m}$ denotes the masking location matrix, and $\boldsymbol{V}\sim N(\mu, \sigma^2)$ denotes the masking Gaussian noise. Ultimately, we obtain the second kind of augmented instance $G^o_{p.2}$=($\mathcal{V}$, $\mathcal{E}$, S$'$) of the original malware instance G_o.

3.3.3 Meta-graph-guide Sampling Attack. An effective method to generate augmented instances is sampling. Different from the sampling-based data augmentation on homogeneous graphs (*e.g.*, ego-net sampling [63] or uniform sampling [59]), we design a tailored meta-graph-guide sampling attack for heterogeneous graphs, which follows certain semantics to sample from the global heterogeneous graph. Specifically, as illustrated in Figure 5(c), given the pre-defined meta-graph M_i and G_o, the meta-graph sampled neighborhood $N^{(i)}$ is:

$$N^{(i)} \;=\; \{Nei|(Nei, Tar) \in M_i, (Tar, Nei) \in M_i\}, \tag{6}$$

where $N^{(i)}$ contains all visited neighbor nodes *Nei* when the target process node *Tar* walks along with meta-graph M_i. Eventually, we obtain the third kind of augmented instance $G^o_{p.3}$= ($N^{(i)}$, $\mathcal{E}^{(i)}$, S$^{(i)}$) of the original few-shot malware instance G_o.

3.3.4 Direct System Calls Attack. To evade sandbox surveillance, more and more malware uses direct system calls to evade API hooks. In general, new malware variants generated by such methods do not call the APIs inside ntdll.dll, so the sandboxes or monitors fail to perceive the malicious activity [18]. To accomplish this, we propose a direct system call attack to generate a practical augmented instance for few-shot malware. Concretely, we remove some key APIs (e.g., NtAdjustPrivilegesToken, NtWriteVirtualMemory, NtDeleteValueKey, and so on) from the original malware behavior events to perform a direct system call attack. In this way, we obtain the fourth kind of augmented instance $G^o_{p.4} = (\mathcal{V} - \mathcal{V}_{api}, \mathcal{E} \subseteq (\mathcal{V} - \mathcal{V}_{api}) \times (\mathcal{V} - \mathcal{V}_{api}), S_{\mathcal{V}-\mathcal{V}_{api}})$ of the original few-shot malware instance G_o.

3.3.5 Obfuscation Attack. Another variant generation method is obfuscation attack, which employs code obfuscation techniques to conceal the original malicious behavior with semantically equivalent but different behavior, thus automatically mistaking similar malware samples within one family for samples from a different family [6]. For example, the malicious API operation CopyFileEx shown in Figure 5(e) can be replaced with several benign or sensitive API operations, including NtOpenFile, NtReadFile, and NtWriteFile.

Specifically, inspired by [34], we propose two obfuscation attacks to generate practical augmented instances for few-shot malware: (1) replace an API call sequence with its equivalent, and (2) insert redundant data flow dependent API calls. In addition, our presented obfuscation attacks can ensure that the graph editing distance [7] between the original malware instance and the augmented instance is as large as possible without affecting the semantics, which is more conducive to promoting the performance of contrastive learning.

Formally, the distance between the original malware instance G_o and the fifth kind of augmented instance $G^o_{p.5} = (\mathcal{V} + \mathcal{V}_{api}, \mathcal{E} \subseteq (\mathcal{V} + \mathcal{V}_{api}) \times (\mathcal{V} + \mathcal{V}_{api}), S_{\mathcal{V}+\mathcal{V}_{api}})$ is defined as

$$d(G_o, G^o_{p.5}) = 1 - \frac{|mcs(G_o, G^o_{p.5})|}{max(|G_o|, |G^o_{p.5}|)}, \tag{7}$$

where $mcs(G_o, G^o_{p.5})$ denotes the maximal common sub-graph of G_o and $G^o_{p.5}$. $|G_o|$ and $|G^o_{p.5}|$ represent the number of nodes in G_o and $G^o_{p.5}$, respectively.

Ultimately, Figure 5 shows five types of augmented instances of the original malware instance; A2-CLM combines each augmented instance and the original instance into a positive pair, which takes the form $(G_o, G^o_{p.m})$. Moreover, A2-CLM randomly chooses the software instance (e.g., benign software) from the rest of the software types as the negative pair, which forms $(G_o, G^o_{q.n})$.

3.4 Contrastive Learning-based Malware Detection

A properly contrastive discriminator will act on multiple positive and negative instances pairs, which detects few-shot malware by gradually learning the "distinguishable" information of different instance pairs in a self-supervised manner. Notably, to answer ***Issue***$_3$, A2-CLM contrasts the graph-level representations of various instance pairs. As shown in Figure 4(c), A2-CLM holds out to capture the similarity between the original malware instance G_o and the adversarial augmented instance G_p, simultaneously, capture the dissimilarity between G_o and the sampled negative instance G_q.

Concretely, in this subsection, we first employ graph attention networks (GATs)[51], a powerful graph neural network that has been shown to be superior to GCN [26] and GraphSAGE [21], to learn the graph-level representations of each pair of contrastive instances. Formally, given the original malware instance G_o, m adversarial augmented positive instances set G^o_P, and n sampled negative instances set G^o_Q, we implement three graph encoders, GAT_o, GAT_p, and GAT_q, to generate the comprehensive representations of the contrastive instance pairs through the following steps:

1) Aggregate node-level representation $\mathbf{h}_{Tar}$: We first acquire the node-level representations $\mathbf{h}_{Tar}$ of the target process node Tar in G_o by iteratively aggregating its own features with those of its important neighbors. Formally, the attention weight $\alpha^{(i)}_{Tar,Nei}$ of the neighbor node Nei can be defined as:

$$\alpha^{(i)}_{Tar,Nei} = \frac{exp(LeakyReLU(\mathbf{W}^T[X_{Tar}, X_{Nei}] + \mathbf{b}))}{\sum_{Nei' \in N^{(i)}_{Tar}} exp(LeakyReLU(\mathbf{W}^T[X_{Tar}, X_{Nei}] + \mathbf{b}))}, \tag{8}$$

where X_{Tar} and X_{Nei} are the feature vectors of node Tar and Nei. $N^{(i)}_{Tar}$ is the neighborhood of the process node Tar guided by meta-graph M_i. Then, the k-th layer of the node-level aggregator is:

$$\mathbf{h}^{(i)(k)}_{Tar} = MLP^{(k)}((1+\epsilon^{(k)})\mathbf{h}^{(i)(k-1)}_{Tar} + \sum_{Nei \in N^{(i)}_{Tar}} \alpha^{(i)}_{Tar,Nei}\mathbf{h}^{(i)(k-1)}_{Nei}), \tag{9}$$

where $k \in \{1, 2, \ldots, K\}$ denotes the index of the layer, $\boldsymbol{h}^{(i)(k-1)}_{Tar}$ and $\boldsymbol{h}^{(i)(k-1)}_{Nei}$ are the node-level representations of target process node Tar and corresponding neighbor node Nei at the (k-1)-th layer, respectively. ϵ_k is a trainable balance parameter. Hence, the full node-level representation of Tar guided by meta-graph M_i is:

$$\mathbf{h}^{(i)}_{Tar} = CONCAT([\mathbf{h}^{(i)(k)}_{Tar}]^K_{k=1}). \tag{10}$$

2) Aggregate graph-level representation $\boldsymbol{h}_{G_o}$: We then aspire to aggregate different node representations into graph embedding space. Similarly, to evaluate the significance of different meta-graphs, we insist on computing the meta-graph attention weight θ_i of each M_i and obtaining the comprehensive graph-level representation $\boldsymbol{h}_{G_o}$ of the target few-shot malware. Formally,

$$\theta_i = \frac{exp(\sigma(\mathbf{b}[\mathbf{W}_b\mathbf{h}^{(i)}_{Tar} \| \mathbf{W}_b\mathbf{h}^{(j)}_{Tar}]))}{\sum_{g \in |M|} exp(\sigma(\mathbf{b}[\mathbf{W}_b\mathbf{h}^{(i)}_{Tar} \| \mathbf{W}_b\mathbf{h}^{(g)}_{Tar}]))}, \tag{11}$$

$$\mathbf{h}_{G_o} = \sum_{i=1}^{|M|} \theta_i \times \mathbf{h}^{(i)}_{Tar}, \tag{12}$$

where $i \neq j \in \{1, \ldots, |M|\}$, and σ is the activation function. $\boldsymbol{b}$ is the weight vector from the input layer to the hidden layer of the neural network, and $\boldsymbol{W}_b$ is the corresponding weight matrix.

Empirically, we implement a non-linear projection head [10] to the graph-level representation $\boldsymbol{h}_{G_o}$ before computing the pair-wise similarity, which is calculated by:

$$\mathbf{z}_{G_o} = MLP(\mathbf{h}_{G_o}). \tag{13}$$

Ultimately, we keep a mini-batch B_o of n negative instances $(G^o_{q.1}, \cdots, G^o_{q.n})$ drawn at random from the rest of the software families and a positive instance G^o_p. From a dictionary look-up perspective, we aim to identify the unique pair of positive instances (denoted by (G_o, G^o_p)) in B_o, and the loss function InfoNCE [36] is defined as:

$$l_{o,p} = \frac{exp(sim(\mathbf{z}_{G_o}, \mathbf{z}_{G^o_p})/\tau)}{exp(sim(\mathbf{z}_{G_o}, \mathbf{z}_{G^o_p})/\tau) + \sum_{d=1}^{n} exp(sim(\mathbf{z}_{G_o}, \mathbf{z}_{G^o_{q.d}})/\tau)}, \tag{14}$$

Algorithm 1 The Overall Procedure of A2-CLM

```
Input: A sensitivity heterogeneous graph of the target malware
    instance_o: $G_o$=($\boldsymbol{A}_o$, $\boldsymbol{S}_o$), Adversarial augmentations: $\mathcal{H}^{PGD}$,
    $\mathcal{H}^{mask}$, $\mathcal{H}^{sample}$, $\mathcal{H}^{DSC}$, $\mathcal{H}^{obfuscation}$, Encoders: $GAT_o$,
    $GAT_p$, $GAT_q$, Projection heads: $MLP_o$, $MLP_p$, $MLP_q$, $m$, $n$;
Output: Trained $GAT_o$, $GAT_p$, $GAT_q$, $MLP_o$, $MLP_p$, $MLP_q$;
 1: $\mathbf{G_P^o}$ = ($G^o_{p.1}$, $G^o_{p.2}$, $\cdots$, $G^o_{p.m}$) = $\mathcal{H}^{PGD}(G_o)$ ∪ $\mathcal{H}^{mask}(G_o)$
    ∪ $\mathcal{H}^{sample}(G_o)$ ∪ $\mathcal{H}^{DSC}(G_o)$ ∪ $\mathcal{H}^{obfuscation}(G_o)$;
 2: $G_Q^o$ = ($G^o_{q.1}$, $G^o_{q.2}$, $\cdots$, $G^o_{q.n}$) = sampling the negative instances
    from the rest of software families;
 3: for $\{G^o_{p.a}\}^m_{a=1} \in \mathbf{G}^o_P$ do
 4:     $B_o = G^o_{p.a} \cup \mathbf{G}^o_Q$;
 5:     $\mathbf{h}_{G_o} = GAT_o(G_o)$;
 6:     $\mathbf{z}_{G_o} = MLP_o(\mathbf{h}_{G_o})$;
 7:     $\mathbf{h}_{G^o_{p.a}} = GAT_p(G^o_{p.a})$;
 8:     $\mathbf{z}_{G^o_{p.a}} = MLP_p(\mathbf{h}_{G^o_{p.a}})$;
 9:     for $\{G^o_{q.d}\}^n_{d=1} \in B_o \backslash \{G^o_{p.a}\}$ do
10:         $\mathbf{h}_{G^o_{q.d}} = GAT_q(G^o_{q.d})$;
11:         $\mathbf{z}_{G^o_{q.d}} = MLP_q(\mathbf{h}_{G^o_{q.d}})$;
12:     end for
13:     Computing $l_{o,p.a}$ by using Eq. 14;
14: end for
15: for $a$ = 1 to $m$ do
16:     $\mathcal{L}=\frac{1}{m}\sum^m_{a=1} l_{o,p.a}$;
17: end for
18: Updating $I_o$ by maximizing $\mathcal{L}$ with backpropagation;
19: $I_p=\lambda_1 * I_p+(1-\lambda_1)*I_o$;
20: $I_q=\lambda_2 * I_q+(1-\lambda_2)*I_o$;
21: return $GAT_o$, $GAT_p$, $GAT_q$, $MLP_o$, $MLP_p$, $MLP_q$
```

where $\boldsymbol{z}_{G_o}$, $\boldsymbol{z}_{G^o_p}$, and $\boldsymbol{z}_{G^o_{q.d}}$ denote the low-dimensional representations of G_o, G^o_p, and $G^o_{q.d}$ after GAT encoders and non-linear projection heads, respectively. τ denotes a preset temperature parameter. Hence, the final loss L is computed across all positive and negative instance pairs, and it is formalized as:

$$\mathcal{L} = \frac{1}{m}\sum^m_{p=1} l_{o,p}. \tag{15}$$

The contrastive loss is then used to train the graph encoders GAT$_o$, GAT$_p$, GAT$_q$, as well as the projection heads MLP$_o$, MLP$_p$, MLP$_q$. Inspired by the momentum strategy [24], we tactfully fine-tune the parameters I_o of GAT$_o$ or MLP$_o$ by backpropagation and then momentum-based update the rest parameters of GAT$_p$ (or MLP$_p$) and GAT$_q$ (or MLP$_q$) by utilizing the I_o when training multiple graph encoders and projection heads. The pseudocode of A2-CLM is depicted in Algorithm 1.

4 EXPERIMENTS

In this section, we conduct ample experiments to show the effectiveness of A2-CLM on sufficient malware detection tasks and few-shot malware detection tasks. We first introduce the four datasets used for experiments and the experiment setup. Then, we demonstrate the experimental results, including the comparison of performance, parameter sensitivity, unknown malware detection, ablation study, and robustness against packed malware.

4.1 Dataset and Baseline Methods

4.1.1 Dataset. We evaluate A2-CLM on four real-world malware benchmark datasets, including *BIG 2015* [41], *Ember* [2], *API Call Sequences* [42], and *ACT-KingKong* [29], which are widely used in recent malware detection research. Detailed statistics are shown in Table 2, and the detailed descriptions are given as follows:

- **BIG 2015** contains 10,868 malware files from 9 malware families and various features (such as function calls, strings, etc.) extracted from the binary by the IDA disassembler tool.
- **Ember** is a realistic dataset with 50% malicious software and 50% benign software that contains eight groups of raw features from 140,000 PE files.
- **API Call Sequences** includes the first 100 API sequences of 37,784 malware samples and 1,079 benign samples that are collected in a sandboxed environment, which is constantly used in Kaggle competitions and dynamic detection research.
- **ACT-KingKong** contains execution reports of 8,494 malicious files and 3,042 benign files from Mar 2019 to Oct 2019, and each report contains 8 types of malware objects, including processes, files, networks, memories, registries, systems, mutexes, and attributes. To identify few-shot malware from different families, we submitted the hash value of each file to VirusTotal[1] and counted 102 distinct malware families.

For all datasets, we randomly divided all categories into 6:2:2 for the training set, validation set, and test set, respectively.

4.1.2 Baseline Methods. We make comparisons with the following state-of-the-art baselines:

- **MalConv** [40] is a CNN-based detection model that can learn the spatial features between the byte sequences of malware.
- **CNN+BPNN** [64] is a hybrid framework that employs CNN and BPNN to extract opcode features and API features.
- **MatchGNet** [55] is a GNN-based model that characterizes the execution events of malware into a heterogeneous graph and extracts metapath-based features to detect malware.
- **MG-DVD** [29] is a dynamic detection framework that incrementally learns graph embedding by utilizing the overlapping information of adjacent sliding windows.
- **API+AAE** [37] is a generative model that proposes an adversarial auto-encoder (AAE) for malware detection.
- **GraphCL** [63] is a pre-training framework that uses subgraph instance discrimination to distinguish between similar and dissimilar few-shot instances.
- **GACL** [47] is a GCN-based model that adopts adversarial learning and contrastive learning.

4.2 Experimental Settings

We train A2-CLM on a machine with a 16 cores Intel(R) Core(TM) i7-6700 CPU @3.40 GHz with 64 GB RAM and 4×NVIDIA Tesla K80 GPU. All of the experiments developed with Python 3.6 are executed on the TensorFlow-GPU framework supported by Ubuntu

[1]https://www.virustotal.com.

Table 2: Statistics of The Four Datasets

Dataset	Samples Distribution									Network Size	
BIG 2015	Ramnit	Lollipop	Kelihos_ver3	Vundo	Simda	Tracur	Kelihos_ver1	Obfuscator.ACY	Gatak	Avg. # Node	Avg. # Degree
	1,541	2,478	2,942	475	42	751	398	1,228	1,013	153.3	1.04
API Call Sequences	Trojan	Downloader	Virus	Spyware	Adware	Dropper	Worm	Backdoor	Benign	Avg. # Node	Avg. # Degree
	12,824	6,560	5,522	5,897	4,449	945	808	779	1,079	84.1	1.29
Ember	Malware	Benign	-	-	-	-	-	-	-	Avg. # Node	Avg. # Degree
	69,860	70,140	-	-	-	-	-	-	-	16.8	1.84
ACT-KingKong	Trojan	Virus	Worm	Backdoor	Downloader	Ransom	Dropper	Benign	-	Avg. # Node	Avg. # Degree
	4,536	1,606	842	660	394	338	118	3,042	-	23.7	3.61
(# Family)	42	17	20	12	4	3	4	1	-	-	-

Table 3: Performance on Malware Detection

Method	**ACT-KingKong dataset**				**Ember dataset**				**BIG 2015 dataset**				**API Call Sequences dataset**			
	Recall	Precision	ACC	F1-score	Recall	Precision	ACC	F1-score	Recall	Precision	ACC	F1-score	Recall	Precision	ACC	F1-score
MalConv [40]	83.32	90.85	84.31	86.92	90.32	92.68	91.55	91.48	95.95	97.28	96.15	96.61	80.58	81.19	80.64	80.88
CNN+BPNN [64]	87.36	91.95	88.29	89.60	92.88	93.52	93.23	93.20	97.26	97.94	97.31	97.60	83.71	84.09	83.73	83.90
MatchGNet [55]	91.37	93.72	91.71	92.53	94.03	96.48	95.35	95.24	97.51	98.02	97.55	97.76	86.82	87.13	86.85	86.97
MG-DVD [29]	96.51	98.16	97.63	97.33	94.87	97.02	95.94	95.93	97.83	98.44	97.83	98.13	86.97	88.28	86.99	87.62
API+AAE [37]	90.28	91.93	90.29	91.10	92.95	95.29	94.16	94.11	96.15	96.71	96.18	96.43	82.66	84.07	83.97	83.36
GraphCL [63]	94.46	94.96	94.47	94.71	93.76	94.83	93.97	94.29	97.54	98.06	97.58	97.80	85.23	85.54	85.23	85.38
GACL [47]	97.35	97.73	97.68	97.54	96.62	97.56	96.69	97.09	98.22	98.43	98.21	98.32	88.15	88.72	88.19	88.43
A2-CLM (ours)	**98.87**	**99.04**	**98.87**	**98.95**	**99.21**	**99.66**	**99.28**	**99.43**	**99.69**	**99.85**	**99.70**	**99.77**	**94.22**	**95.06**	**94.25**	**94.64**
% Improvement	1.52%	1.31%	1.19%	1.41%	2.59%	2.10%	2.59%	2.34%	1.47%	1.42%	1.49%	1.45%	6.07%	6.34%	6.06%	6.21%

16.0.4 operating system. We measure the performance of A2-CLM in terms of recall, precision, ACC, F1-score, and AUC.

We utilize Adam [25] for optimization with the learning rate of 0.005, decay of 0.00001, mini-batch size of 64, temperature $\tau = 0.07$, embedding size of 128, number of layers of GAT to 4, and momentum λ_1=λ_2= 0.99.

4.3 Few-shot Malware Detection Results

4.3.1 Effectiveness Evaluation. In this subsection, we evaluate the detection performance of A2-CLM on the sufficient malware samples tasks as well as the few-shot malware tasks. We run our proposed framework 10 times and report the average results in Table 3 and Table 4. The experimental results show that our proposed A2-CLM outperforms all baseline methods in terms of all evaluation metrics, especially in the 1-shot task and 10-shot task. In fact, the improvement of A2-CLM can be attributed to the following traits:

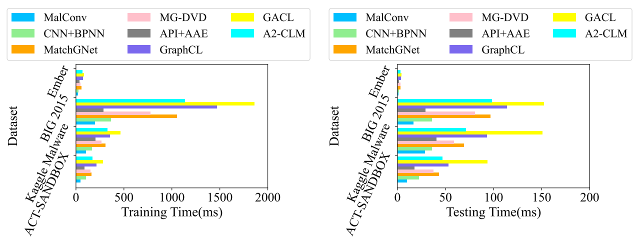
First, compared with the feature-based supervised learning detection methods, such as MalConv and CNN+BPNN, our proposed A2-CLM achieves more than 2.39%~10.58% improvement in terms of ACC on the sufficient malware samples tasks and more than 33.87%~47.62% improvement in terms of ACC on the 1-shot malware tasks. These experimental results mainly contribute to the fact that A2-CLM not only models the various malware objects and their interactions into heterogeneous graphs to capture the context structure semantics of few-shot malware but also employs graph attention networks (GATs) to generate the fine-grained graph-level representations, which is beneficial to improving the detection performance of A2-CLM.

Second, on sufficiently large malware sample tasks, the detection performances of the existing graph-based supervised learning malware detection methods, such as MatchGNet and MG-DVD, are close to that of A2-CLM. This is because they ingeniously leverage graph neural networks (GNNs) to capture the high-order semantic information from a large quantity of malware samples, which can better express the evolving patterns of the malware. They do poorly on 1-shot tasks, however, because they rely on a large corpus of labeled samples to train the model in a supervised manner. Differently, our proposed A2-CLM no longer concentrates on labeling information but is equipped with the capability to learn more discriminative and robust few-shot malware fingerprints by self-supervised contrastive learning.

Third, compared with the most relevant self-supervised works involving API+AAE, GraphCL, and GACL, the advantages of A2-CLM are twofold. On the one hand, API+AAE only extracts the isolated API features, and its detection performance heavily depends on the generated adversarial samples and whether they contain sufficient detail. Contrarily, A2-CLM is capable of recognizing few-shot malware by investigating the approximate discriminative patterns based on the semantically rich sensitivity heterogeneous graph instances. On the other hand, although GraphCL and GACL

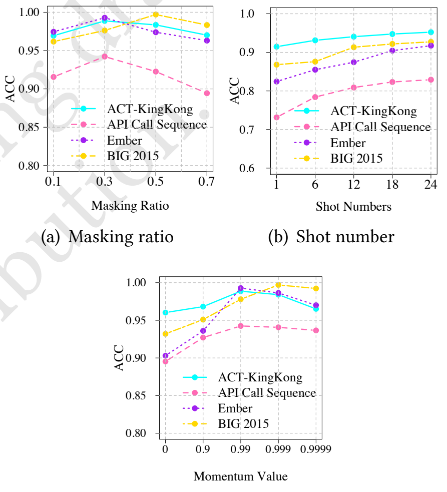
Table 4: Performance on Few-shot Malware Detection

Method	ACT-KingKong dataset				Ember dataset				BIG 2015 dataset				API Call Sequences dataset			
	1-shot		10-shot		1-shot		10-shot		1-shot		10-shot		1-shot		10-shot	
	ACC	F1-score	ACC	F1-score	ACC	F1-score	ACC	F1-score	ACC	F1-score	ACC	F1-score	ACC	F1-score	ACC	F1-score
MalConv [40]	38.24	38.71	47.36	47.42	34.55	34.59	42.16	42.66	37.64	38.09	45.88	45.85	32.86	32.90	45.37	45.63
CNN+BPNN [64]	43.85	43.89	53.05	53.11	35.72	35.78	44.37	44.36	40.91	41.28	49.63	49.61	39.28	39.75	51.66	51.71
MatchGNet [55]	50.12	50.58	57.19	57.15	40.31	40.42	50.08	50.23	46.29	46.25	53.18	53.57	47.43	47.45	57.19	57.28
MG-DVD [29]	52.50	52.59	61.77	61.73	44.27	44.71	51.82	52.09	48.53	48.50	55.92	55.95	48.52	48.69	60.74	60.72
API+AAE [37]	64.63	64.89	69.43	69.42	50.68	50.72	55.23	55.21	60.74	60.68	64.51	64.88	60.94	61.14	69.25	69.24
GraphCL [63]	67.71	67.86	71.28	71.35	60.83	61.59	64.75	64.77	68.16	68.19	71.35	71.41	63.66	63.61	70.83	70.85
GACL [47]	80.59	80.63	82.81	82.84	74.65	74.69	78.66	78.61	77.57	77.72	80.24	80.25	68.57	68.62	77.06	77.12
A2-CLM (ours)	**91.47**	**91.54**	**93.74**	**93.89**	**82.44**	**82.48**	**87.19**	**87.15**	**86.81**	**86.79**	**89.47**	**89.52**	**73.15**	**73.22**	**80.58**	**80.65**



(a) Training Time (b) Testing Time

Figure 6: Time efficiency comparison on four datasets.



(a) Masking ratio (b) Shot number

(c) Momentum value

Figure 7: Performance change of A2-CLM with major parameters.

can provide promising detection frameworks in a self-supervised contrastive learning, they only investigate oversimplified data augmentations for homogeneous graphs because they ignore nonlinear heterogeneous dependencies and lack practical malware data augmentation, degrading the few-shot malware detection performance. On the contrary, A2-CLM designs five insightful adversarial heterogeneous graph augmentations that make full use of adversarial attacks and expert knowledge, enhancing the effectiveness of few-shot malware detection.

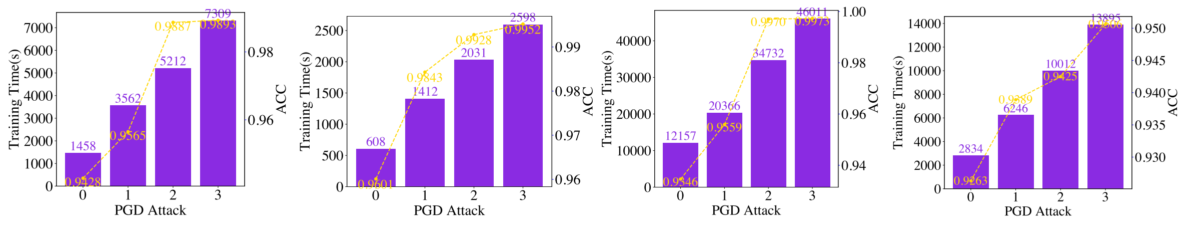
4.3.2 Efficiency Evaluation. Here we study the time efficiency of A2-CLM processing each sample on four datasets. The comparison results are shown in Figure 6. We make two crucial observations.

First, from Figures 6(a-b), we can see that A2-CLM is consistently faster than the existing homogeneous graph contrastive learning models (i.e., GraphCL and GACL) over all datasets. The reason for efficiency improvement is attributed to A2-CLM utilizing meta-graph structures to guide the walk and generate positive instances rather than uniformly sampling a lot of noise in GraphCL and GACL, which is capable of reducing the walking cost and training cost. *Second*, Figures 6(a-b) also show the time overhead of the feature-based detection methods (i.e., MalConv, CNN+BPNN, and API+AAE) is significantly less than that of graph-based methods (i.e., MatchGNet, MG-DVD, GraphCL, GACL, and A2-CLM) in all datasets because they do not need to consider the complex graph structure involved with various types of malware entities and interactive relationships, which empirically proves that the simpler the model, the lower the time complexity.

4.4 Parameter Sensitivity

In this subsection, we investigate the sensitivity of four major parameters on the performance of A2-CLM, including the masking ratio of $\boldsymbol{L}_m$ (*i.e.*, r), the shot number of samples (*i.e.*, *n-shot*), the number of PGD attacks (*i.e.*, k), and the momentum value (*i.e.*, λ).

4.4.1 Effect of Masking Ratio r. We first explore the significance of attribute masking ratio r for the A2-CLM framework. As depicted in Figure 7(a), with the masking ratio r growing, the ACC value first steadily rises, which can be attributed to masking more values in the sensitivity attribute matrix, which would produce more robust positive instances for few-shot malware contrastive learning. Additionally, with the masking ratio growing, the improvement of ACC in Figure 7(a) is within a certain range, which is related to the



(a) ACT-KingKong dataset (b) Ember dataset (c) BIG 2015 dataset (d) API Call Sequence dataset

Figure 8: The performance of A2-CLM with different numbers of PGD attacks.

Table 5: Performance on Unknown Malware Detection

Method	D_{known}=$D_{unknown}$=50% of the software family				
	Recall	Precision	ACC	F1-score	AUC
MalConv [40]	75.61	75.98	75.68	75.94	73.32
CNN+BPNN [64]	79.64	80.35	79.73	79.99	77.54
MatchGNet [55]	85.25	85.92	85.25	85.58	82.91
MG-DVD [29]	86.44	87.15	86.53	86.79	84.87
API+AAE [37]	87.28	88.10	87.34	87.69	85.95
GraphCL [63]	90.67	90.92	90.68	90.79	88.03
GACL [47]	92.06	93.13	92.11	92.59	89.28
A2-CLM	**95.19**	**95.74**	**95.37**	**95.46**	**93.30**

sparsity of the attribute matrix of the samples, so we experimentally select the masking ratio r at 30% for effectiveness consideration.

4.4.2 Effect of The Shot Number n. We also investigate the impact of shot number n on the performance of A2-CLM. Figure 7(b) presents the detection accuracy of A2-CLM when gradually increasing the shot number n on different datasets. With more samples included in each type of all datasets, the accuracy increases obviously, as expected. Particularly, there is a 3.76%, 5.91%, 9.76%, and 10.29% increment of accuracy when the shot number is increased from 1 to 10 in the ACT-KingKong dataset, BIG 2015 dataset, Ember dataset, and API Call Sequence dataset, respectively, which attributes to more shots included in each type and means the proposed A2-CLM can learn more distinguishable information from more instances to help distinguish new malware. In addition, as the shot number increases, the detection performance of A2-CLM becomes more and more gentle, indicating that A2-CLM can get very excellent performance by adding a small number of samples.

4.4.3 Effect of The Momentum Value λ. We further study the effect of the different momentum values in the proposed A2-CLM framework. Momentum λ plays an important role in training efficient contrastive learning. Figure 7(c) shows the detection accuracy with different momentum values on four datasets. It is worth noting that a larger momentum value brings better performance, which shows that the more stable and consistent the evolution of λ_1 and λ_2, the better the performance. As shown in Figure 7(c), we can find that an appropriate value can achieve the desired performance. Specifically, the best performance is reached when λ=0.99 on the API Call Sequence dataset, Ember dataset, and ACT-KingKong dataset.

4.4.4 Effect of The Number of PGD Attacks k. The PGD attacks have a strong influence on generating realistic and robust contrastive instances. As shown in Figures 8(a-d), the ACC increases significantly as k increases, which can be attributed to more PGD attacks; more attribute-level adversarial instances can be obtained. However, the higher the number of PGD attacks, the higher the training cost. That is, the A2-CLM's training time almost doubles for each additional PGD attack. As a result, we use two PGD attacks to maintain the trade-off between A2-CLM's effectiveness and efficiency.

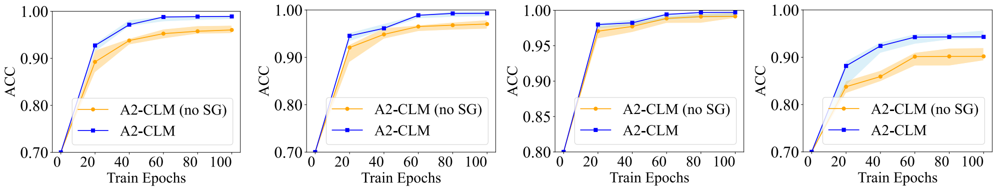
4.5 Unknown Malware Detection

This subsection focuses on evaluating A2-CLM for unknown malware detection. To simulate unknown malware instances in the wild or unseen new families, we randomly split all samples from the ACT-KingKong dataset into two sets: D_{known} = 50% of the software families (i.e., 52 families), $D_{unknown}$ = the remaining 50% of the software families (i.e., 51 families). Moreover, to ensure that each family has been in the unknown set as far as possible, we employ k-fold cross-validation to evaluate our A2-CLM and report the average experimental results.

We report the average detection performance of all 8 methods on the unknown malware detection task in Table 5. All evaluation metrics of A2-CLM outperform the state-of-the-art methods. Concretely, the accuracy of A2-CLM is at least 3.26% better than baseline methods, which is attributed to A2-CLM leveraging adversarial augmentation-based contrastive learning to perform well on unknown families by capturing the matching patterns and mismatching patterns via its intrinsic discriminative mechanism. On the contrary, existing supervised learning methods, especially feature-based detection methods, suffer a significant drop in detection performance when faced with unknown families or unknown malware because they strongly rely on known fingerprints and blacklists, which cannot handle samples that are outside the training set.

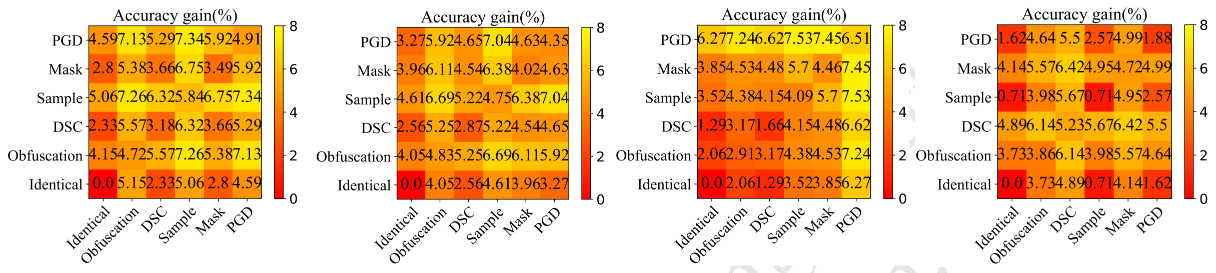
4.6 Ablation Study

4.6.1 Sensitivity Grading Method. We first study the contribution of the sensitivity grading method on A2-CLM. As shown in Figure 9, the detection performance of A2-CLM is slightly superior to that of its variant model, A2-CLM no SG, on all datasets, confirming that



(a) ACT-KingKong dataset (b) Ember dataset (c) BIG 2015 dataset (d) API Call Sequence dataset

Figure 9: The effect of the sensitivity grading method on A2-CLM.



(a) ACT-KingKong dataset (b) Ember dataset (c) BIG 2015 dataset (d) API Call Sequence dataset

Figure 10: Accuracy gain(%) of A2-CLM when contrasting different adversarial heterogeneous graph augmentation pairs on all datasets. "Identical" stands for a no-augmentation. The lighter color indicates better performance gains.

the security semantics implicit in run-time behavior are beneficial to boosting the detection capability of A2-CLM. Furthermore, the varying degrees of sensitivity allow the few-shot malware detection results to be interpreted.

4.6.2 Adversarial Heterogeneous Graph Augmentation. We then assess the role of different adversarial heterogeneous graph augmentations in our A2-CLM framework. As illustrated in Figures 10(a-d), we can discover that without any data augmentation in the contrastive learning, it is not helpful for the downstream few-shot malware detection tasks, judging from the value 0 of the "Identical" pair. In contrast, from the top rows or the right columns in Figures 10(a-d), we believe that composing different adversarial heterogeneous graph augmentations can enhance the detection performance. Concretely, when we composed "Sample" and "PGD", the maximum accuracy gains were 7.34%, 7.04%, and 7.53% for the ACT-KingKong dataset, Ember dataset, and BIG 2015 dataset, respectively. Similarly, when we composed "DSC" and "Mask", the maximum accuracy gain was 6.42% for the API Call Sequence dataset. These findings prove that applying the same type of heterogeneous graph data augmentations does not bring out the best detection performance; however, by composing different types of adversarial heterogeneous graph augmentations, especially at the structure-level and attribute-level, the proposed A2-CLM can reach the best detection performance, which avoids the learned representations overfitting.

4.6.3 Meta-graph. Finally, we investigate the impact of various meta-graphs on the performance of few-shot malware detection by gradually incorporating meta-graphs into our A2-CLM. In Figure 11, we can observe that by incorporating more meta-graphs, the ACCs of A2-CLM on all datasets are higher, which empirically

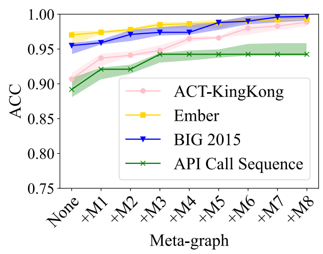


Figure 11: Performance change of A2-CLM when gradually incorporating meta-graphs in terms of ACC.

proves that diverse meta-graphs are capable of capturing the unique semantic information and helping capture the intrinsic malicious patterns of few-shot malware. Particularly, we can find that when adding M_1, M_4, and M_6, A2-CLM has a significant boost in the ACT-KingKong dataset; a similar situation happens when we add M_1 and M_3 in the API Call Sequence dataset. Additionally, the detection performance has a slight improvement in the BIG 2015 dataset and the Ember dataset, which is attributed to the fact that the malware heterogeneous graph instances in these two datasets have fewer structures that satisfy the pre-defined meta-graphs, making it difficult to reflect the advantages of meta-graphs in A2-CLM.

4.7 Packed Malware

Using the ACT-SANDBOX dataset's encountered packed malware, we tested A2-CLM's robustness against packers. As shown in Table 6, we consider four types of packers, whose complexity ranges

Table 6: Performance on Packed Malware Detection

Packer	Packer Type	# Malware	ACC
UPX	Type-I	176	0.989
BobSoft Mini Delphi	Type-I	73	0.977
ASPack	Type-III	84	0.974
Armadillo	Type-VI	138	0.956

from Type-I to Type-VI [16]. We see that A2-CLM can unpack and analyze samples packed with the common packers (i.e., UPX, BobSoft Mini Delphi, ASPack, and Armadillo), regardless of the Type-I packers that can be easily unpacked only using a single unpacking routine or the Type-VI packers that can be unpacked using a multi-layer routine. Hence, we believe that A2-CLM is robust enough to enable a large-scale study of packed malware. However, some packers use virtualization [50] or repackaging [28, 43] technologies to generate complex variants by converting software into bytecode and evading sandbox detection. Consequently, A2-CLM cannot handle the virtualization and repackaging of malware variants, which account for a tiny fraction of packed malware [50].

5 RELATED WORK

5.1 Malware Detection

The research on malware detection mostly focuses on supervised learning, which leverages deep learning and even graph neural networks to extract various static or dynamic fingerprint features of malware. Their successes depend on handling a large number of labeled samples. Concretely, Tian *et al.* [49] first adopted structured heterogeneous information networks (HINs) for malware detection and then designed a similarity-based approach to detect the malware. Wang *et al.* [55] designed MatchGNet, where the similarities between the target malware and all benign samples can be measured by leveraging meta-path-based graph representations. Recently, Liu *et al.* [29] attempted to investigate the real-time detection framework based on the sliding window, which can effectively and efficiently defend against malware attacks with two dynamic walk-based heterogeneous graph learning methods.

However, the aforementioned methods not only ignore the security semantics implicit in run-time behavioral parameters but also heavily depend on a large corpus of labeled samples, limiting the capability of few-shot malware detection.

5.2 Few-shot Malware Classification

Research on few-shot malware classification is still in its infancy. Bai *et al.* [5] adopted a siamese-network method to identify the few-shot Android malware, which extracted three types of syntactic features and trained a multi-layer perceptron (MLP). Wang *et al.* [53] proposed a few-shot malware dynamic analysis approach based on meta-learning, which has the ability to classify novel malware families that have never met. To solve the one-shot malware outbreak, Park *et al.* [37] developed the generative adversarial auto-encoder (AAE) [33] for malware detection.

However, the aforementioned few-shot malware classification methods have certain flaws. Firstly, the meta-learning-based classification method is essential to hold the memory of the previously learned samples, which undoubtedly causes catastrophic forgetting and new data over-fitting; Secondly, the generative model heavily concentrates on each detail of the sample, which would cause poor detection performance once the generated adversarial sample did not contain enough information for matching. Conversely, our proposed A2-CLM designs heterogeneous graph contrastive learning to conquer the aforementioned limitations in few-shot malware classification, which is capable of recognizing new few-shot malware by investigating only the approximate discriminative patterns instead of sufficient detail.

5.3 Graph Contrastive Learning

Graph contrastive learning has been successfully applied to many tasks [31, 59]. For example, DGI [52] first attempted to extend the Deep InfoMax in CV to the graph data, which implemented a graph convolutional network (GCN) as the encoder and maximized the mutual information between input and output. Unlike DGI, which targets learning node-level representation, Sun *et al.* [45] intended to maximize the mutual information between graph-level representations. Inspired by what CMC has done to improve Deep InfoMax, Hassani *et al.* [23] presented a contrastive multi-view representation learning method for graph data that adopts graph diffusion to yield positive sample pairs. Qiu *et al.* [39] pioneered the use of instance discrimination for graph pre-training. They utilized random walks with restart (RWR) to generate independent subgraphs as instances and calculated the InfoNCE [36] loss, which is friendly to large-scale graphs. You *et al.* [63] believed that node neighborhood reconstruction is a local-global contrast and that emphasizing neighborhood information over structural information would destroy structural information. Therefore, they proposed four augmentation methods based on homogeneous graphs to produce the positive samples, which achieve better performance on dissimilar datasets.

However, the existing data augmentation of the aforementioned graph contrastive learning methods emphasizes generating augmented instances for homogeneous graphs, which are feeble for malware heterogeneous graphs due to their neglecting the heterogeneous dependencies of various malware entities. To address this problem, our A2-CLM proposes to generate more practical contrastive pairs by investigating five types of adversarial heterogeneous graph augmentation, which is beneficial to improving the model's robustness and enhancing the effectiveness of few-shot malware detection.

6 CONCLUSION

In this paper, we propose a novel few-shot malware detection model named A2-CLM, which overcomes the label scarcity issue by implementing self-supervised graph contrastive learning. A2-CLM combines sensitivity heterogeneous graphs and adversarial data augmentations to detect malware efficiently and robustly. Our evaluation has verified the superiority of our proposed A2-CLM in identifying few-shot malware tasks.

REFERENCES

[1] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. 2020. VisualPhishNet: Zero-Day Phishing Website Detection by Visual Similarity. In *Proceedings of the*

2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20). 1681–1698.
[2] Hyrum S Anderson and Phil Roth. 2018. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637* (2018).
[3] Anshul Arora, Sateesh Kumar Peddoju, and Mauro Conti. 2020. PermPair: Android Malware Detection Using Permission Pairs. *IEEE Trans. Inf. Forensics Secur.* 15 (2020), 1968–1982. https://doi.org/10.1109/TIFS.2019.2950134
[4] AvTest. 2020. Malware Statistics. https://www.av-test.org/en/statistics/malware/.
[5] Yude Bai, Zhenchang Xing, Xiaohong Li, Zhiyong Feng, and Duoyuan Ma. 2020. Unsuccessful story about few shot malware family classification and siamese network to the rescue. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1560–1571.
[6] Jean-Marie Borello and Ludovic Mé. 2008. Code obfuscation techniques for metamorphic viruses. *Journal in Computer Virology* 4, 3 (2008), 211–220.
[7] Horst Bunke and Kim Shearer. 1998. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters* 19, 3-4 (1998), 255–259.
[8] Tanmoy Chakraborty, Fabio Pierazzi, and VS Subrahmanian. 2017. Ec2: Ensemble clustering and classification for predicting android malware families. *IEEE Transactions on Dependable and Secure Computing* 17, 2 (2017), 262–277.
[9] Li Chen, Mingwei Zhang, Chih-Yuan Yang, and Ravi Sahita. 2017. POSTER: semi-supervised classification for dynamic android malware detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. 2479–2481.
[10] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. 1597–1607.
[11] Xiaohui Chen, Zhiyu Hao, Lun Li, Lei Cui, Yiran Zhu, Zhenquan Ding, and Yongji Liu. 2022. CruParamer: Learning on Parameter-Augmented API Sequences for Malware Detection. *IEEE Transactions on Information Forensics and Security* 17 (2022), 788–803.
[12] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555* (2020).
[13] Daniele Cono D'Elia, Emilio Coppa, Federico Palmaro, and Lorenzo Cavallaro. 2020. On the Dissection of Evasive Malware. *IEEE Trans. Inf. Forensics Secur.* 15 (2020), 2750–2765. https://doi.org/10.1109/TIFS.2020.2976559
[14] Yujie Fan, Shifu Hou, Yiming Zhang, Yanfang Ye, and Melih Abdulhayoglu. 2018. Gotcha-sly malware! scorpion a metagraph2vec based malware detection system. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 253–262.
[15] Ruitao Feng, Sen Chen, Xiaofei Xie, Guozhu Meng, Shang-Wei Lin, and Yang Liu. 2021. A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices. *IEEE Trans. Inf. Forensics Secur.* 16 (2021), 1563–1578. https://doi.org/10.1109/TIFS.2020.3025436
[16] Jonathan Fuller, Ranjita Pai Kasturi, Amit Sikder, Haichuan Xu, Berat Arik, Vivek Verma, Ehsan Asdar, and Brendan Saltaformaggio. 2021. C3PO: Large-Scale Study of Covert Monitoring of C&C Servers via Over-Permissioned Protocol Infiltration. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3352–3365.
[17] José Gaviria de la Puerta and Borja Sanz. 2017. Using Dalvik opcodes for malware detection on Android. *Logic Journal of the IGPL* 25, 6 (2017), 938–948.
[18] H. Gavriel. 2018. Malware Mitigation When Direct System Calls Are Used. https://www.cyberbit.com/blog/endpoint-security/malware-mitigation-when-direct-system-calls-are-used/.
[19] Samujjwal Ghosh and Maunendra Sankar Desarkar. 2018. Class Specific TF-IDF Boosting for Short-text Classification: Application to Short-texts Generated During Disasters. In *Companion of the The Web Conference 2018 on The Web Conference 2018, WWW 2018, Lyon , France, April 23-27, 2018*, Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, 1629–1637. https://doi.org/10.1145/3184558.3191621
[20] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. 2020. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733* (2020).
[21] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1025–1035.
[22] Weijie Han, Jingfeng Xue, Yong Wang, Lu Huang, Zixiao Kong, and Limin Mao. 2019. MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *computers & security* 83 (2019), 208–233.
[23] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*. PMLR, 4116–4126.
[24] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9729–9738.
[25] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[26] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[27] Lingpeng Kong, Cyprien de Masson d'Autume, Wang Ling, Lei Yu, Zihang Dai, and Dani Yogatama. 2019. A mutual information maximization perspective of language representation learning. *arXiv preprint arXiv:1910.08350* (2019).
[28] Li Li, Tegawendé F Bissyandé, and Jacques Klein. 2019. Rebooting research on detecting repackaged android apps: Literature review and benchmark. *IEEE Transactions on Software Engineering* 47, 4 (2019), 676–693.
[29] Chen Liu, Bo Li, Jun Zhao, Ming Su, and Xu-Dong Liu. 2021. MG-DVD: A Real-time Framework for Malware Variant Detection Based on Dynamic Heterogeneous Graph Learning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. 1512–1519.
[30] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. 2019. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*. 1777–1794.
[31] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. 2021. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[32] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).
[33] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. 2015. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644* (2015).
[34] Jiang Ming, Zhi Xin, Pengwei Lan, Dinghao Wu, Peng Liu, and Bing Mao. 2017. Impeding behavior-based malware analysis via replacement attacks to malware specifications. *Journal of Computer Virology and Hacking Techniques* 13, 3 (2017), 193–207.
[35] Zequn Niu, Jingfeng Xue, Dacheng Qu, Yong Wang, Jun Zheng, and Hongfei Zhu. 2022. A novel approach based on adaptive online analysis of encrypted traffic for identifying Malware in IIoT. *Information Sciences* 601 (2022), 162–174.
[36] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).
[37] Sean Park, Iqbal Gondal, Joarder Kamruzzaman, and Leo Zhang. 2019. One-Shot Malware Outbreak Detection using Spatio-Temporal Isomorphic Dynamic Features. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 751–756.
[38] Razvan Pascanu, Jack W Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. 2015. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1916–1920.
[39] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1150–1160.
[40] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. 2018. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.
[41] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. 2018. Microsoft malware classification challenge. *arXiv preprint arXiv:1802.10135* (2018).
[42] Angelo Schranko de Oliveira and Renato José Sassi. 2019. Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks. (2019).
[43] Luman Shi, Jiang Ming, Jianming Fu, Guojun Peng, Dongpeng Xu, Kun Gao, and Xuanchen Pan. 2020. Vahunt: Warding off new repackaged android malware in app-virtualization's clothing. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. 535–549.
[44] Software. 2017. Chinese Academy of Sciences releases "King Kong" malware intelligent analysis system. *Dual-use technologies and products* (2017). https://doi.org/10.19385/j.cnki.1009-8119.2017.11.046
[45] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2019. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000* (2019).
[46] Mingshen Sun, Xiaolei Li, John CS Lui, Richard TB Ma, and Zhenkai Liang. 2016. Monet: a user-oriented behavior-based malware variants detection system for android. *IEEE Transactions on Information Forensics and Security* 12, 5 (2016), 1103–1112.
[47] Tiening Sun, Zhong Qian, Sujun Dong, Peifeng Li, and Qiaoming Zhu. 2022. Rumor Detection on Social Media with Graph Adversarial Contrastive Learning. In *Proceedings of the ACM Web Conference 2022*. 2789–2797.
[48] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. 2021. Adversarial graph augmentation to improve graph contrastive learning. *Advances in Neural Information Processing Systems* 34 (2021), 15920–15933.

[49] Ke Tian, Danfeng Yao, Barbara G Ryder, Gang Tan, and Guojun Peng. 2017. Detection of repackaged android malware with code-heterogeneity features. *IEEE Transactions on Dependable and Secure Computing* 17, 1 (2017), 64–77.

[50] Xabier Ugarte-Pedrero, Davide Balzarotti, Igor Santos, and Pablo G Bringas. 2015. SoK: Deep packer inspection: A longitudinal study of the complexity of run-time packers. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 659–673.

[51] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *stat* 1050 (2017), 20.

[52] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep graph infomax. *arXiv preprint arXiv:1809.10341* (2018).

[53] Peng Wang, Zhijie Tang, and Junfeng Wang. 2021. A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling. *Computers & Security* 106 (2021), 102273.

[54] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A Gunter, et al. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis.. In *NDSS*.

[55] Shen Wang and S Yu Philip. 2019. Heterogeneous Graph Matching Networks: Application to Unknown Malware Detection. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 5401–5408.

[56] Xiao Wang and Guo-Jun Qi. 2022. Contrastive learning with stronger augmentations. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[57] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)* 53, 3 (2020), 1–34.

[58] Ho Chung Wu, Robert Wing Pong Luk, Kam-Fai Wong, and Kui-Lam Kwok. 2008. Interpreting TF-IDF term weights as making relevance decisions. *ACM Trans. Inf. Syst.* 26, 3 (2008), 13:1–13:37. https://doi.org/10.1145/1361684.1361686

[59] Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. 2021. Self-supervised learning of graph neural networks: A unified review. *arXiv preprint arXiv:2102.10757* (2021).

[60] Wenhan Xiong, Jingfei Du, William Yang Wang, and Veselin Stoyanov. 2019. Pretrained encyclopedia: Weakly supervised knowledge-pretrained language model. *arXiv preprint arXiv:1912.09637* (2019).

[61] Zhicong Yan, Jun Wu, Gaolei Li, Shenghong Li, and Mohsen Guizani. 2021. Deep Neural Backdoor in Semi-Supervised Learning: Threats and Countermeasures. *IEEE Trans. Inf. Forensics Secur.* 16 (2021), 4827–4842. https://doi.org/10.1109/TIFS.2021.3116431

[62] Jianhua Yin and Jianyong Wang. 2014. A dirichlet multinomial mixture model-based approach for short text clustering. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani (Eds.). ACM, 233–242. https://doi.org/10.1145/2623330.2623715

[63] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems* 33 (2020), 5812–5823.

[64] Jixin Zhang, Zheng Qin, Hui Yin, Lu Ou, and Kehuan Zhang. 2019. A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding. *Computers & Security* 84 (2019), 376–392.

[65] Jixin Zhang, Kehuan Zhang, Zheng Qin, Hui Yin, and Qixin Wu. 2018. Sensitive system calls based packed malware variants detection using principal component initialized MultiLayers neural networks. *Cybersecurity* 1, 1 (2018), 1–13.

[66] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. 2020. Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security (CCS '20)*. 757–770.

[67] Zhilu Zhang and Mert Sabuncu. 2018. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems* 31 (2018).

[68] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-Graph Based Recommendation Fusion over Heterogeneous Information Networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 635–644. https://doi.org/10.1145/3097983.3098063

[69] Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo. 2021. Structural Attack against Graph Based Android Malware Detection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*. 3218–3235.