Malicious Repositories Detection with Adversarial Heterogeneous Graph Contrastive Learning

Yiyue Qian
University of Notre Dame
Notre Dame, Indiana, USA
yqian5@nd.edu

Yiming Zhang
Case Western Reserve University
Cleveland, Ohio, USA
yxz2092@case.edu

Nitesh Chawla
University of Notre Dame
Notre Dame, Indiana, USA
nchawla@nd.edu

Yanfang Ye*
University of Notre Dame
Notre Dame, Indiana, USA
yye7@nd.edu

Chuxu Zhang*
Brandeis University
Waltham, Massachusetts, USA
chuxuzhang@brandeis.edu

ABSTRACT

GitHub, as the largest social coding platform, has attracted an increasing number of cybercriminals to disseminate malware by posting malicious code repositories. To address the imminent problem, some tools were developed to detect malicious repositories based on the code content. However, most of them ignore the rich relational information among repositories and usually require abundant labeled data to train the model. To this end, one effective way is to exploit unlabeled data to pre-train a model which considers both structural relation and code content of repositories, and further transfer the pre-trained model to the downstream tasks with labeled repository data. In this paper, we propose a novel model *adversarial contrastive learning on heterogeneous graph* (**CLA-HG**) to detect malicious repository in GitHub. First of all, CLA-HG builds a heterogeneous graph (HG) to comprehensively model repository data. Afterwards, to exploit unlabeled information in HG, CLA-HG introduces a dual-stream graph contrastive learning mechanism that distinguishes both adversarial subgraph pairs and standard subgraph pairs to pre-train graph neural networks using unlabeled data. Finally, the pre-trained model is fine-tuned to the downstream malicious repository detection task enhanced by a knowledge distillation (KD) module. Extensive experiments on two collected datasets from GitHub demonstrate the effectiveness of CLA-HG in comparison with state-of-the-art methods and popular commercial anti-malware products.

CCS CONCEPTS

• **Security and privacy** → **Social network security and privacy**; • **Computing methodologies** → *Machine learning algorithms*; • **Networks** → Online social networks.

KEYWORDS

Malicious repository detection; Heterogeneous graph; Graph neural network; Self-supervised learning

ACM Reference Format:
Yiyue Qian, Yiming Zhang, Nitesh Chawla, Yanfang Ye, and Chuxu Zhang. 2022. Malicious Repositories Detection with Adversarial Heterogeneous Graph Contrastive Learning. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557384

*Corresponding authors.

CIKM '22, October 17–21, 2022, Atlanta, GA, USA.

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557384

1 INTRODUCTION

As of November 2021, with more than 73 million registered developers and over 200 million repositories [44], GitHub has become the largest social coding platform where it enables global developers to host open-source projects and develop software collaboratively. Due to high openness and convenience, developers can easily upload and download public projects for program development. Meanwhile, GitHub has also attracted an increasing number of cybercriminals to perform cyberattacks and disseminate malware by aggressively uploading malicious repositories. Once repositories with malicious code are downloaded by legitimate developers, the attacked equipment will be infiltrated and compromised, which will cause huge losses to individuals and organizations. Hence, destroying the dissemination of malware is very urgent. To this end, in this paper, we develop a novel model to detect malicious repositories on GitHub.

To maintain safe and productive social coding platforms, some methods [8, 28, 32] have been proposed to analyze the malicious activities in GitHub. However, these methods merely analyze the code content but ignore the rich relational information among nodes in graphs, which is not sufficient enough to address the security problem in GitHub. For instance, GitHub offers two vulnerability detection products LGMT [35] and CodeQL [34] to generate automatic code vulnerability reports based on the code content. However, such kinds of security policies are very limited because tens of thousands of malicious repositories still can be detected. Although some recent methods [32, 59] have improved the performance in detecting malicious activities, they require a lot of time and resources to obtain sufficient labeled repository data yet ignore the valuable and handy unlabeled information. For example, SourceFinder [32] labeled almost ten thousand repositories to

train a supervised classifier but the classification performance is not promising enough.

Therefore, in this paper, we develop a novel framework called CLA-HG (Figure 1) which not only considers code content information and rich relations in GitHub but also exploits the handy unlabeled data for malicious repository detection. Specifically, we first construct a heterogeneous graph (HG) to depict the code content of repositories and relations among entities (e.g., repository and user) in GitHub. Instead of directly utilizing graph neural networks (GNN) to fuse the content information and structural relations, we develop a self-supervised module to pre-train an expressive GNN encoder with unlabeled data. In particular, we introduce a dual-stream graph contrastive learning mechanism that defines heterogeneous subgraph pairs discrimination as the contrastive learning task. To obtain a better pre-trained encoder, we leverage adversarial attacks to generate more challenging positive and more effective negative pairs in heterogeneous subgraph contrastive learning. Thus, our proposed dual-stream graph contrastive learning framework aims to distinguish adversarial subgraph instance pairs (adversarial stream) and standard subgraph instance pairs (standard stream). Finally, the pre-trained model is fine-tuned to the downstream malicious repository detection task, where the optimization process is further augmented with a knowledge distillation module. To conclude, the major contributions of our work include:

- To solve the imminent problem of malicious repositories detection, we develop a novel model called CLA-HG which comprehensively models content information and structural relations as well as unlabeled data in GitHub.
- To the best of our knowledge, CLA-HG is among the earliest work integrating contrastive learning and adversarial training on graph to generate challenging samples for representation learning.
- We collect one general dataset from GitHub and conduct comprehensive experiments. The results demonstrate the effectiveness of CLA-HG by comparison with state-of-the-art methods and commercial anti-malware products.

2 RELATED WORK

This work is closely related to the studies of graph neural networks, Contrastive learning on graph, adversarial learning on graph, and malicious repository detection.

Graph Neural Networks. Our graph repesentation learning is inspired by graph neural networks (GNNs). Existing GNNs can be generally divided into two streams, spectral-based GNNs [2, 5, 10] and spatial-based GNNs [9, 24, 40, 45, 55]. Spectral-based GNNs aim to present nodes in graph and perform convolution in the spectral space. For instance, Henaff et al. extended the spectral networks to learn the node embedding in graph [10]. Spatial-based GNNs usually consider the relational structure information between nodes and aggregate the information of nodes based on the local structural information. For example, GCN [18] implements layer-wise propagation rule to learn the node embedding. In this paper, we employ GCN as our base model to learn the node representation.

Contrastive Learning on Graph. Existing contrastive learning on graph models [31, 37, 46, 52, 53, 62] usually train an encoder to learn the graph representation by discriminating positive pairs and negative pairs (e.g., subgraph) generated from graph. For instance, GCC [31] proposes the subgraph instance discrimination as the contrastive learning task to learn the graph representation. In addition, GraphCL [52] introduces four types of graph augmentations to generate graphs in different views to conduct contrastive learning. In heterogeneous domain, HeCo [43] employs a cross-view (schema view and meta-path view) contrastive mechanism to learn node embedding. The performance of graph contrastive learning relies heavily on the quality of positive pairs and negative pairs instances. Hence, to compensate for the above limitation, we aim to devise a graph contrastive learing framework by generating more effective positive pairs and challenging negative pairs.

Adversarial Learning on Graph. Motivated by the strength of adversarial learning on image and text, some recent works have started to study adversarial learning on graphs to enhance the network robustness in supervised learning [14, 58, 60]. For instance, Zhang et al. proposed GNNGUARD which absorbs the adversarial perturbations using Gaussian distributions as node representations in each layer of the network [58]. Recently, some works [15, 52] extend to improve the model robustness with adversarial transformations during self-supervised learning. For instance, GROC [15] aims to build a contrastive learning algorithm with adversarial attacks to improve the network robustness. In addition, some recent works implement adversarial attacks to generate powerful contrastive pairs for graph contrastive leanrning. For examples, GASSL [50] leverages adversarial attacks to inject perturbations to graph features for generating more challenging contrastive pairs, and further train the graph encoder. AD-GCL considers adversarial attacks as a graph augmentation strategy to automatically learn useful data augmentations for graph dataset. However, the aforementioned works focus on homogeneous graphs and ignore the heterogeneity property in HG. In addition, existing works directly apply adversarial attacks to graph may be too risky and overloaded for graph encoders. Accordingly, we leverage adversarial attacks to generate more challenging positive and negative pairs for heterogeneous graph contrastive learning. To alleviate the workload of graph encoder, we propose dual-stream heterogeneous graph contrastive learning to train graph encoders over both adversarial subgraph instance pairs and standard subgraph instance pairs.

Malicious Repository Detection. Most existing studies about malicious repository detection mainly focus on analyzing the code content [8, 32, 51]. For example, La et al. [19] analyzed the content information of PE file in repository to detect malicious repositories in GitHub. However, these methods ignore the rich relational structure information among entities in social coding platforms. Besides, some existing works focus on collecting and utilizing labeled data but ignore the valuable unlabeled information [29, 59]. For instance, GitCyber [59] integrates both code-based content and relational structure information in GitHub but ignores the valuable unlabeled information among collected data. Unlike existing works, we aim to integrate relational structural information, code content, and unlabeled data to detect malicious repositories in social coding platforms.

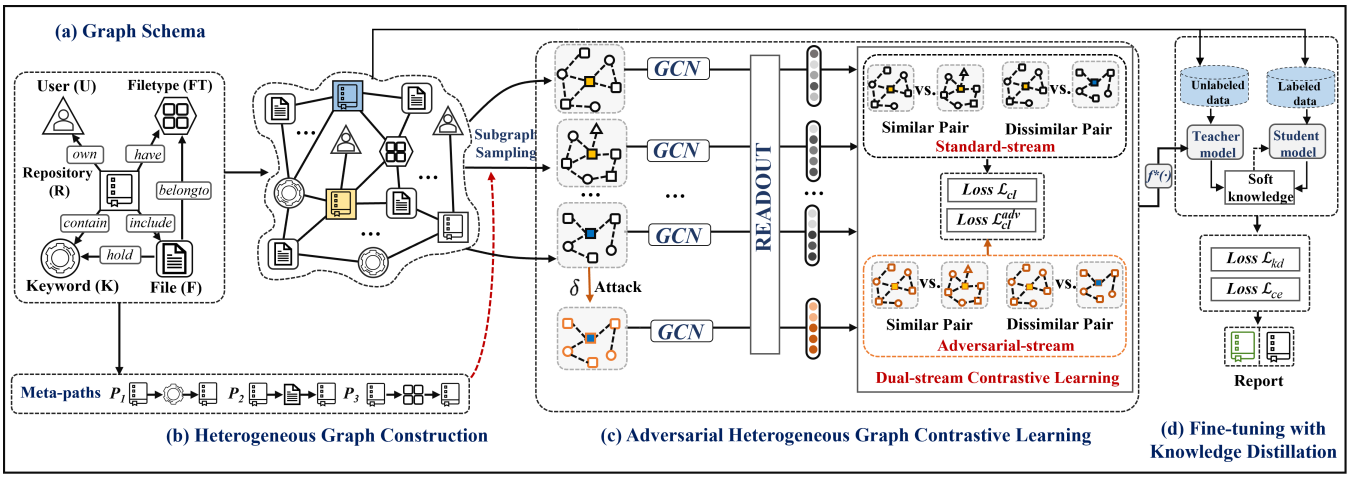

Figure 1: The overall framework of CLA-HG: (a) It defines five types of entities and six kinds of relationships among entities; (b)Then it constructs a HG based on the graph schema to depict data in GitHub; (c) Afterwards, it employs adversarial attacks to sample adversarial heterogeneous subgraphs for generating more challenging positive and negative pairs and further train GCN via optimizing standard-stream and adversarial-stream contrastive losses. (d) The pre-trained model $f^*(\cdot)$ from (c) is fine-tuned to the downstream task with labeled repository data and the optimization process is augmented by a knowledge distillation module.

3 PRELIMINARY

In this section, we introduce three related definitions of this paper (i.e., heterogeneous graph, meta-path, graph convolutional networks, and adversarial training) and further define the problem of malicious repository detection.

Definition 3.1. **Heterogeneous Graph.** To comprehensively depict the repository data, we build a heterogeneous graph (HG) [20, 21, 49], $G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$, where $\mathcal{V}$ is the set of different types of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathcal{X}$ is the attributed feature set. They are associated with a node type mapping function $\phi : \mathcal{V} \rightarrow \mathcal{T}$ and an edge type mapping function $\phi : \mathcal{E} \rightarrow \mathcal{R}$, where $\mathcal{T}$ and $\mathcal{R}$ is the set of node types and the set of relation types with $|\mathcal{T}| + |\mathcal{R}| > 2$. The graph schema for G is a graph with nodes from $\mathcal{V}$ and edges from $\mathcal{E}$. As we can see from Figure 1.(a), there are five types of nodes (i.e., repository, user, keyword, file, and filetype) and six types of relations (e.g., repository-contain-keyword) in HG.

Definition 3.2. **Meta-Path.** A meta-path [56, 57] $\mathcal{P}$ is a path defined on the graph schema, which is denoted in the form of $T_1 \xrightarrow{R_1} T_2 \xrightarrow{R_2} \ldots \xrightarrow{R_L} T_{L+1}$ where $R = R_1 \cdot R_2 \cdot \ldots \cdot R_L$ ($T_i \in \mathcal{T}$) is the composite relation between node types T_1 and T_{L+1}, and L is the length of $\mathcal{P}$. Figure 1.(b) shows three meta-paths (i.e., P_1, P_2, P_3) manually defined by semantic relations among different types of entities based on domain knowledge.

Definition 3.3. **Graph Convolutional Network.** As GCN [18] is powerful to learn node representation by considering both structure of graph G and node features $\mathcal{X}$, we employ GCN as the encoder in this paper. It is a layer-wise propagation rule-based model to learn the node embedding $h_i \in \mathbb{R}^d$ (d: embedding dimension) corresponding to repository node v_i. In particular, GCN is formulated as follows:

$$H^{l+1} = \sigma(\widetilde{A} H^l W^l), \tag{1}$$

where H^{l+1} denotes the node representations at $l+1$ layer and $H^0 = X$ represents the original attribute feature of the node. $\widetilde{A}$ is a symmetric normalization of A with self-loop, i.e., $\widetilde{A} = \hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}}$ with $\hat{A} = A + I_N$. A, I_N, $\hat{D}$ are the adjacency matrix, the identity matrix, and the diagonal node degree matrix of $\hat{A}$ respectively. W^l denotes the weight matrix at l-th layer, and σ is the activation function. For simplicity, we use $\mathbf{h} = \mathrm{GCN}(X, A)$ to denote the GCN model and $\mathbf{h}$ is the node embedding.

Definition 3.4. **Adversarial Training.** Inspired by the power of adversarial training, we aim to utilize adversarial perturbation to facilitate the contrastive learning by generating more challenging positive pairs and negative pairs, which is formulated as:

$$\delta = \underset{\|\delta'\|_\infty \leq \epsilon}{\arg\max}\, L(\theta, X + \delta'), \tag{2}$$

where δ is the adversarial perturbation, X is the node feature, θ is the model parameters and L is the loss with parameter θ. Adversarial training tries to solve the following optimization problem:

$$\min_{\theta} \mathbb{E}\,(L(\theta, X + \delta)). \tag{3}$$

Problem 1. ***Malicious Repository Detection.*** *Given a HG $G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ along with repository labels $Y = (y_1, \ldots, y_N)$ ($y_i = 1$ means malicious and $y_i = 0$ means benign), we aim to build a self-supervised learning model over unlabeled data, which can be fine-tuned to the downstream malicious repositories detection task with labeled data.*

4 METHODOLOGY

In this section, we present the details of CLA-HG (Figure 1) which includes three key steps: (1) heterogeneous graph construction on GitHub data (Figure 1.(b)); (2) adversarial heterogeneous graph contrastive leaning for model pre-training (Figure 1.(c)); (3) model fine-tuning with knowledge distillation (Figure 1.(d)).

4.1 Heterogeneous Graph Construction

To identify whether a repository in GitHub is malicious or benign, we consider both the semantic content of repositories and the rich relations among entities for characterizing repositories comprehensively. The content-based features and rich structural relations are described below.

4.1.1 Content-based Features. For each repository, we extract five related elements (considered as five types of entities in HG), including repository content, corresponding user profile, keywords in the repository, file, and file type, to effectively describe the repository. To obtain the feature vector of repository, we extract and combine the information of introduction and description, and then apply *BERT* [6] to convert the text information to a fixed-length feature vector. For the corresponding user, we extract the user profile and also convert it to a feature vector. Besides, each repository has several files and each file has the corresponding file type (e.g., .exe, .C, and .java). Hence, we encode the file name and the type of file as the feature vector of file and file type, respectively. To obtain the keyword feature vector, we first extract a set of keywords from all repositories and implement CHI statistic [54] to select effective keywords by considering the level of independence among keywords and the class label. To conclude, we acquire 30,000 effective keywords for GitHub-Malware dataset and 52,530 effective keywords for GitHub-Corona dataset (introduced in Section 5.1). After keyword extraction, each selected keyword is considered as a node in HG and we implement BERT to acquire the feature vectors.

4.1.2 Relations. Besides extracting the content information of each entity, we also consider the rich relations among different types of entities, which is also very essential to judge the legitimacy of repositories: (1) *R1*: the *user-own-repository* relation indicates that a user owns or has contribution to a repository; (2) *R2*: the *repository-contain-keyword* relation denotes that a repository contains a specific keyword; (3) *R3*: the *repository-have-filetype* relation shows that a file in a repository has a specific file type; (4) *R4*: the *repository-include-file* relation indicates that a repository includes a specific file; (5) *R5*: the *file-belong-filetype* relation depicts that a file belongs to a specific file type. (6) *R6*: the *file-hold-keyword* relation denotes that a file holds a specific keyword; To better describe the relatedness over repositories, we manually define three meta-paths (i.e., P_1, P_2, P_3 in Figure 1) to extract the rich connection among different type of entities. For example, P_1: repository $\xrightarrow{R_2}$ keyword $\xrightarrow{R_2^{-1}}$ repository denotes two repositories are connected if they contain the same keyword.

To summarize, as the graph schema shown (Figure 1), we build a HG by integrating both semantic content of repositories and rich structure relation among different entities (i.e., R1-R6 among five types of nodes). Each entity is attached with the feature vector.

Table 1: Content and relation information from GitHub.

Content Feature	
Repository	title, topic, program language, read.me
User	user name, bio info, bio description
Keyword	keyword extracted from code files
File	filename, file description, SHA-1
File type	types of files (e.g., exe file, C file, and java file)
Relation	
R1: user-own-repository	R2: repository-contain-keyword
R3: repository-have-filetype	R4: repository-include-file
R5: file-belong-filetype	R6: file-hold-keyword

4.2 Adversarial Heterogeneous Graph Contrastive Learning

Motivated by the recent success of graph contrastive learning [31, 52] which implements different data augmentations to generate contrastive instances, we propose a novel heterogeneous graph contrastive learning model with adversarial perturbation to generate more challenging positive pairs and negative pairs so that we can obtain a better pre-trained model for node representations. As shown in Figure 1.(c), CLA-HG has four consecutive components: adversarial heterogeneous subgraph sampling, adversarial subgraph encoding, embedding projection, and dual-stream graph contrastive learning.

Adversarial Heterogeneous Subgraph Sampling. Most existing graph contrastive learning methods employ random walk, node dropping, and edge dropping/adding to sample subgraphs [31, 52]. However, these methods ignore the rich heterogeneous structural relations in HG. To this end, we propose a hybrid strategy to sample heterogeneous subgraphs by considering the rich heterogeneous structure information. Specifically, given a set of root nodes $\mathcal{S}_q$ and meta-paths set $\mathcal{P}$, we aim to generate heterogeneous subgraph $G_q^{\mathcal{P}} = (\mathcal{V}_q, \mathcal{E}_q, \mathcal{X}_q)$, where $\mathcal{V}_q$ is the node set of $\mathcal{S}_q$ including all m-hop node neighbors; $\mathcal{E}_q \subseteq \mathcal{V}_q \times \mathcal{V}_q$ is the edge set; $\mathcal{X}_q$ is the attribute feature set of all nodes in $\mathcal{V}_q$. For each node v_i in $\mathcal{S}_q$, we employ random walk with restarted guided by $\mathcal{P}$ and obtained a set of nodes $S_i^{\mathcal{P}}$. After that, we adopt a hybrid sampling strategy to sample fixed-size neighbors. If the neighbor number of node v_i in $S_i^{\mathcal{P}}$ is less than the fixed-size number, we first sample neighbors in $S_i^{\mathcal{P}}$ without replacement and then sample neighbors out of $S_i^{\mathcal{P}}$. Otherwise, we only sample neighbors in $S_i^{\mathcal{P}}$ to generate heterogeneous subgraphs. By repeating the previous two steps, we obtain a pair of heterogeneous subgraph instance $(G_q^{\mathcal{P}}, G_k^{\mathcal{P}})$ which can be regarded as a similar pair if they are generated by the same root nodes ($\mathcal{S}_q = \mathcal{S}_k$). Otherwise they are considered as dissimilar instance pair ($\mathcal{S}_q \neq \mathcal{S}_k$).

In order to generate more challenging positive pairs and negative pairs for the above graph contrastive learning task, we propose to perform adversarial attacks to subgraph pairs. For each subgraph pair $(G_q^{\mathcal{P}}, G_k^{\mathcal{P}})$, based on Eqution 2, we first generate (δ_q, δ_k) as adversarial perturbations for the attribute feature (X_q, X_k) of

subgraph pair, using the PGD attack algorithm [23]. Then the adversarial subgraph pair can be denoted as $(\widetilde{G}_q^{\mathcal{P}}, \widetilde{G}_k^{\mathcal{P}})$, where $\widetilde{G}_q^{\mathcal{P}} = (\mathcal{V}_q, \mathcal{E}_q, \mathcal{X}_q + \delta_q)$ and $\widetilde{G}_k^{\mathcal{P}} = (\mathcal{V}_k, \mathcal{E}_k, \mathcal{X}_k + \delta_k)$.

Adversarial Subgraph Encoding. Given a set of sampled adversarial subgraphs, CLA-HG aims to train an encoder $f(\cdot)$ to learn the representation of subgraphs. In this paper, we apply GCN to learn node embeddings:

$$\widetilde{\mathbf{h}} = \mathrm{GCN}(g(X+\delta), A) = \widetilde{A}\,\mathrm{ReLU}\left(\widetilde{A}(X+\delta)\,W_T\,W^0\right)W^1, \quad (4)$$

where $g(X) = (X+\delta)W_T$ is the function to transform the feature of different types of nodes in $\widetilde{G}^{\mathcal{P}}$ to a common space; After obtaining node embeddings, we employ a READOUT layer to get the graph embedding for $\widetilde{G}^{\mathcal{P}}$:

$$\widetilde{\mathbf{H}} = \mathrm{READOUT}(\widetilde{G}^{\mathcal{P}}) = \sum_{v_i \in \mathcal{V}'} \widetilde{\mathbf{h}}_i, \quad (5)$$

where $\mathcal{V}'$ is the node set of $\widetilde{G}^{\mathcal{P}}$.

Projection. After obtaining the embeddings of adversarial subgraphs, we aim to apply the nonlinear projection heads to convert all subgraph embeddings to a same space for comparison. The projection heads can be formally defined as follows:

$$\widetilde{\mathbf{Z}} = \mathrm{MLP}(\widetilde{\mathbf{H}}). \quad (6)$$

Dual-stream Graph Contrastive Learning. Inspired by SimCLR [3], we define the discrimination of adversarial heterogeneous subgraphs as the contrastive learning task which maximizes the agreements among adversarial subgraph pairs. Specifically, in each epoch, we first randomly sample a mini-batch of n subgraphs and each subgraph is generated by different root node sets $\mathcal{S}$. We then randomly sample another mini-batch of n subgraphs with the same set of root nodes as the previous mini-batch. Two subgraphs with the same root node set can be regarded as positive pair (similar pair) while other $2(n-1)$ subgraph pairs are treated as negative pairs (dissimilar pairs). Afterwards, we inject adversarial perturbations δ into each subgraph. The contrastive loss on adversarial subgraph contrastive learning can be formulated as:

$$\mathcal{L}_{\mathrm{cl}}^{\mathrm{adv}} = \frac{1}{2n}\sum_{p=1}^{n}\left[L(\widetilde{\mathcal{G}}_{2p-1}^{\mathcal{P}}, \widetilde{\mathcal{G}}_{2p}^{\mathcal{P}}) + L(\widetilde{\mathcal{G}}_{2p}^{\mathcal{P}}, \widetilde{\mathcal{G}}_{2p-1}^{\mathcal{P}})\right], \quad (7)$$

$$L(\widetilde{\mathcal{G}}_q^{\mathcal{P}}, \widetilde{\mathcal{G}}_k^{\mathcal{P}}) = -\log\frac{\exp(\mathrm{sim}(\widetilde{\mathbf{Z}}_q, \widetilde{\mathbf{Z}}_k)/\tau)}{\sum_{p=1}^{2n}\mathbb{1}_{[p\neq q]}\exp(\mathrm{sim}(\widetilde{\mathbf{Z}}_q, \widetilde{\mathbf{Z}}_p)/\tau)}, \quad (8)$$

where $\widetilde{\mathcal{G}}_q^{\mathcal{P}}$ and $\widetilde{\mathcal{G}}_k^{\mathcal{P}}$ are in different mini-batchs, $\mathrm{sim}(\widetilde{\mathbf{Z}}_q, \widetilde{\mathbf{Z}}_k)$ is the cosine similarity between the projected embedding of $(\widetilde{\mathcal{G}}_q^{\mathcal{P}}, \widetilde{\mathcal{G}}_k^{\mathcal{P}})$, τ is the temperature parameter, and $\mathbb{1}_{[p\neq q]} \in (0,1)$ is an indicator function, $\mathbb{1} = 1$ when $p \neq q$ otherwise $\mathbb{1} = 0$.

However, performing two different attacks to subgraphs in each pair is too challenging for the encoder $f(\cdot)$ to learn useful representation in contrastive learning. In this case, to alleviate the encoder's workload against adversarial attacks, we introduce a dual-stream graph contrastive learning mechanism that integrates graph contrastive learning on both adversarial subgraph pairs and standard subgraph pairs without adversarial attacks. Accordingly, the dual-stream contrastive loss $\mathcal{L}_{\mathrm{dual}}$ includes the contrastive loss term $\mathcal{L}_{\mathrm{cl}}^{\mathrm{adv}}$ on adversarial subgraph pairs (adversarial-stream) and another contrastive loss term $\mathcal{L}_{\mathrm{cl}}$ on standard subgraphs (standard-stream):

$$\mathcal{L}_{\mathrm{dual}} = \lambda_{adv}\,\mathcal{L}_{\mathrm{cl}}^{\mathrm{adv}} + \mathcal{L}_{\mathrm{cl}}, \quad (9)$$

where λ_{adv} is the trade-off hyper-parameters to control the influence of adversarial attacks. Note that both adversarial branch and standard branch share all weights in encoder $f(\cdot)$.

4.3 Model Fine-tuning

By performing sufficient steps in heterogeneous subgraph contrastive training, we obtain a pre-trained graph encoder $f(\cdot)^*$ to learn the node representation in HG. Different from most existing methods that directly transfer the pre-trained encoder $f(\cdot)^*$ to downstream model [12, 13, 22], we are expired by these works [4, 30, 61] that distills the pre-trained model over unlabeled data can gain some useful information to downstream tasks. Hence, we further introduce the knowledge distillation technique (KD) that distills the unlabeled information (a.k.a. "soft" knowledge) from pre-trained encoder $f^*(\cdot)$ (teacher model) and further transfer the soft knowledge to another model (student model). By mimicking teacher model, student model is able to learn more soft knowledge from unlabeled data which cannot be expressed by the labeled data [4, 47]. In particular, we introduce two types of knowledge distillation (i.e., individual and relational knowledge distillation) for model fine-tuning.

4.3.1 Individual Knowledge Distillation. Individual knowledge usually refers to the individual output of the network layers (intermediate layer or last layer of network). The idea of individual knowledge distillation (IKD) is that student model aims to mimic the individual output information of teacher model [41]. In this paper, we define the individual knowledge as the probability of malicious repository. Specifically, we minimize the following individual knowledge distillation loss between teacher model and student model:

$$\mathcal{L}_{\text{kd-I}} = -t^2\sum_i\sum_{c=0}^{1} P_c^T(z_i,t)\log P_c^S(z_i,t), \quad (10)$$

where $P_c(z_i,t) = \exp(z_i/t)/\sum_c \exp(z_i/t)$. $z_i = \mathrm{logit}(\mathbf{h}_i)$ denotes the logits of node embedding; c is the label of repository node; t is the temperature index to soften the peaky softmax function. The teacher model producing $P_c^T(z_i,t)$ is fixed during the distillation process while the student model producing $P_c^S(z_i,t)$ is trained.

4.3.2 Relational Knowledge Distillation. Relational knowledge distillation (RKD) aims to distill the mutual relation within the output embedding generated by teacher model and further transfer such relational knowledge to student model [26]. Different from individual knowledge distillation, it computes a relational potential function ϕ and then encourages student model to learn the relational function ϕ among repository embedding obtained from teacher model. In this paper, we consider the distance-wise relations as the potential function ϕ and calculate the Euclidean distance among repository data in the embedding space. Specifically, we minimize the following relational knowledge distillation loss:

$$\mathcal{L}_{\text{kd-R}} = \sum_{(v_i,v_o)\in\mathcal{G})} L_h(\phi^T(\mathbf{h}_i,\mathbf{h}_o), \phi^S(\mathbf{h}_i,\mathbf{h}_o)), \quad (11)$$

where $\phi(\mathbf{h}_i, \mathbf{h}_o) = \frac{1}{u}||\mathbf{h}_i - \mathbf{h}_o||^2$, u is the normalization factor for distance, L_h is the loss that penalizes difference between teacher and student model. In this paper, we use the Huber loss to penalize the difference:

$$L_h(a,b) = \begin{cases} \frac{1}{2}(a-b)^2 & |a-b| \le \alpha \\ |a-b| - \frac{1}{2} & \text{otherwise.} \end{cases} \tag{12}$$

α is the controlled hyper-parameter.

4.3.3 Fine-tuning with Knowledge Distillation. Finally, we combine the knowledge distillation loss $\mathcal{L}_{\text{kd}}$ and the malicious repository detection loss (i.e., cross-entropy) $\mathcal{L}_{\text{ce}}$ to fine-tune the model:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{ce}} + \lambda_{kd}\mathcal{L}_{\text{kd}}. \tag{13}$$

where $\mathcal{L}_{\text{kd}} = \mathcal{L}_{\text{kd-I}}$ if we transfer the individual knowledge to student model, otherwise $\mathcal{L}_{\text{kd}} = \mathcal{L}_{\text{kd-R}}$. λ_{kd} is the trade-off weight for balancing two losses. Note that student model has the same structure as the teacher model in this paper. The pseudo-code of CLA-HG is provided in Algorithm 1:

Algorithm 1: Training Procedure of CLA-HG

Data: Heterogeneous graph G, Network backbone $f(\cdot)$
Result: Repository report (malicious or benign)
for *each epoch* **do**
  Sample batch-size sets of root nodes and employ random walk guided by meta-path $\mathcal{P}$ for root nodes.
  Implement hybrid strategy to sample neighbors and generate batch-size pairs of heterogeneous subgraphs $(G_q^{\mathcal{P}}, G_k^{\mathcal{P}})$.
  Initialize the perturbation (δ_q, δ_k) and generate the corresponding adversarial subgraph pairs $(\widetilde{G}_q^{\mathcal{P}}, \widetilde{G}_k^{\mathcal{P}})$.
  Apply $f(\cdot)$ to get the node embedding $\mathbf{h}$ and $\widetilde{\mathbf{h}}$.
  Employ READOUT layer to obtain the subgraph embedding $\mathbf{H}$ and $\widetilde{\mathbf{H}}$ for $G^{\mathcal{P}}$ and $\widetilde{G}^{\mathcal{P}}$ respectively.
  Apply projection heads to project $\widetilde{\mathbf{H}}$ to $\widetilde{\mathbf{Z}}$.
  Optimize $f(\cdot)$ by minimizing $\mathcal{L}_{\text{dual}}$ in Eq. 9.
return $f^*(\cdot)$.
Distill "soft" information from pre-trained encoder $f^*(\cdot)$.
Fine-tune $f^*(\cdot)$ by minimizing $\mathcal{L}_{\text{total}}$ in Eq. 13.

5 EXPERIMENTS

In this section, we introduce two datasets collected from GitHub. Then we conduct extensive experiments to evaluate the performance of CLA-HG. Afterwards, we conduct additional experiments to analyze the effectiveness of each model component, the success of CLA-HG by comparison with commercial anti-malware products, the visualizations of repository representations, and the analysis of hyper-parameters sensitivity.

5.1 Data Collection

In this paper, we leverage two real-world datasets from Github, i.e., GitHub-Malware (collected by this work) and GitHub-Corona (provided by Meta-AHIN [29]) to evaluate CLA-HG.

GitHub-Malware Dataset. Based on a set of malicious related keywords (e.g., spy, mine, and bitcoin), we crawl the repository dataset published from Jan 2021 to Jun 2021, and obtain 8,260 repositories and 7,110 profiles of developers in GitHub. To obtain the ground truth of repository data, we closely follow the two-step mechanism provided by Meta-AHIN [29]. Specifically, we first employ VirusTotal [1] including over 70 anti-malware scanning tools to validate the legitimacy of repositories. We also remove oversize repositories due to the limitation of file size in VirusTotal (i.e., 200 MB). Based on the scanned report from VirusTotal, we then ask anti-malware experts to further analyze suspected repositories with less than 5 malicious reports from VirusTotal. To conclude, we get the final labeled data including 3,956 malicious repositories and 4,304 benign repositories. According to the designed graph schema illustrated in Figure 1, the constructed HG has 55,392 nodes (i.e., 8,260 repository nodes, 7,110 user nodes, 30,000 keyword nodes, 9,283 file nodes, 739 file type nodes) and 4,467,759 edges including relation R1-R6.

GitHub-Corona Dataset. This data is provided by Meta-AHIN [29] which aims to analyze the malicious repository in GitHub during the pandemic. The provided dataset was collected based on a set of COVID-19 related keywords (e.g., coronavirus, pandemic, and vaccine) from Feb 2020 to Dec 2020. To conclude, the GitHub-Corona Dataset has 20,895 repositories including 6,965 malicious repositories and 13,930 benign repositories related to 18,785 users. Moreover, the constructed HG based on COVID-19 related data has 176,035 nodes (i.e., 20,895 repository nodes, 18,785 user nodes, 52,530 keyword nodes, 82,451 file nodes, 1,374 file type nodes) and 5,127,956 edges including relation R1-R6. Note that, as we also consider the type of file as node type in this work, the number of nodes and edegs is slightly different from Meta-AHIN.

5.2 Baseline Methods

To comprehensively evaluate our model CLA-HG, we compare CLA-HG with seventeen baseline models which are divided into four groups: feature-based models (G1), graph-based models (G2), malicious repository detection methods (G3), and graph contrastive learning models (G4).

Feature-based models. We extract a set of effective keywords for each repository and then feed the combined keywords to BERT to get the repository embedding. We then feed the repository embedding to Logistic regression (LR) [11] or a 2-layer DNN [36] to detect malicious repository.

Graph-based models. We implement seven graph learning methods to learn the node representation in graph, and further feed the node embedding into a 2-layer MLP to detect malicious repository. Specifically, we first employ DeepWalk [27] to learn node embedding (ignoring the heterogeneous property and attribute information) by modeling structure proximity. The length of the random walk for each node is set as 40. Also, we implement metapath2vec [7] to learn the semantic information of two defined metapaths [38] (P_1 and P_2) in this application. The walk length of each node is set as 40 as well. In addition, we also implement five graph neural network based representation learning models including GCN [18], GAT [40], GIN [48], R-GCN [33], and HAN [42] to learn the node embedding by leveraging both node feature and graph

Table 2: Performance comparison of all methods with different percentages of training data.

Setting		GitHub-Malware						GitHub-Corona					
		20%		50%		80%		20%		50%		80%	
Group	Model	ACC	REC	ACC	REC	ACC	REC	ACC	REC	ACC	REC	ACC	REC
G1	Feature+LR [11]	0.6483	0.6521	0.7155	0.7175	0.7520	0.7538	0.6368	0.6412	0.7005	0.7016	0.7424	0.7435
	Feature+MLP [36]	0.6665	0.6753	0.7354	0.7366	0.7620	0.7714	0.6426	0.6484	0.7138	0.7125	0.7518	0.7525
G2	Deepwalk [27]	0.7492	0.7503	0.7757	0.7835	0.7811	0.7952	0.7256	0.7266	0.7546	0.7602	0.7723	0.7712
	metapath2vec [7]	0.7539	0.7675	0.7832	0.7925	0.7957	0.8013	0.7388	0.7396	0.7664	0.7687	0.7858	0.7904
	GCN [18]	0.7682	0.7730	0.7865	0.7995	0.8058	0.8053	0.7378	0.7355	0.7672	0.7692	0.7895	0.7918
	GAT [40]	0.7630	0.7755	0.7825	0.7968	0.8045	0.8074	0.7358	0.7386	0.7652	0.7672	0.7992	0.7935
	GIN [48]	0.7726	0.7768	0.7920	0.7934	0.8126	0.8135	0.7425	0.7409	0.7738	0.7682	0.7945	0.7925
	R-GCN [33]	0.7841	0.7852	0.8015	0.8053	0.8221	0.8269	0.7687	0.7635	0.7892	0.7826	0.7997	0.7913
	HAN [42]	0.7889	0.7810	0.8079	0.8002	0.8235	0.8258	0.7698	0.7653	0.7899	0.7833	0.7931	0.7959
G3	SourceFinder [32]	0.6866	0.6725	0.7510	0.7446	0.7698	0.7571	0.6554	0.6583	0.7268	0.7315	0.7627	0.7566
	GitCyber [59]	0.7715	0.7668	0.7876	0.7889	0.8005	0.8066	0.7469	0.7402	0.7685	0.7632	0.7813	0.7887
	Meta-AHIN [29]	0.8159	0.8163	0.8504	0.8417	0.8613	0.8624	0.7902	0.7953	0.8061	0.8124	0.8247	0.8305
G4	GCC [31]	0.8053	0.7982	0.8387	0.8341	0.8523	0.8506	0.7825	0.7853	0.7936	0.8016	0.8125	0.8238
	GraphCL [52]	0.8123	0.8051	0.8405	0.8391	0.8582	0.8604	0.7895	0.7935	0.7955	0.8093	0.8182	0.8296
	HeCo [43]	0.8215	0.8234	0.8524	0.8518	0.8634	0.8657	0.8051	0.8024	0.8135	0.8194	0.8324	0.8345
	GASSL [50]	0.8173	0.8141	0.8562	0.8554	0.8641	0.8634	0.7997	0.7995	0.8104	0.8121	0.8237	0.8321
	AD-GCL [39]	0.8145	0.8114	0.8578	0.8575	0.8687	0.8698	0.8005	0.8017	0.8155	0.8203	0.8385	0.8387
Ours	**CLA-HG (IKD)**	0.8501	0.8524	**0.8791**	**0.8924**	0.8954	0.8947	0.8251	0.8302	**0.8465**	**0.8493**	0.8617	0.8654
	CLA-HG (RKD)	**0.8537**	**0.8567**	0.8769	0.8855	**0.9023**	**0.9017**	**0.8284**	**0.8336**	0.8425	0.8453	**0.8667**	**0.8705**

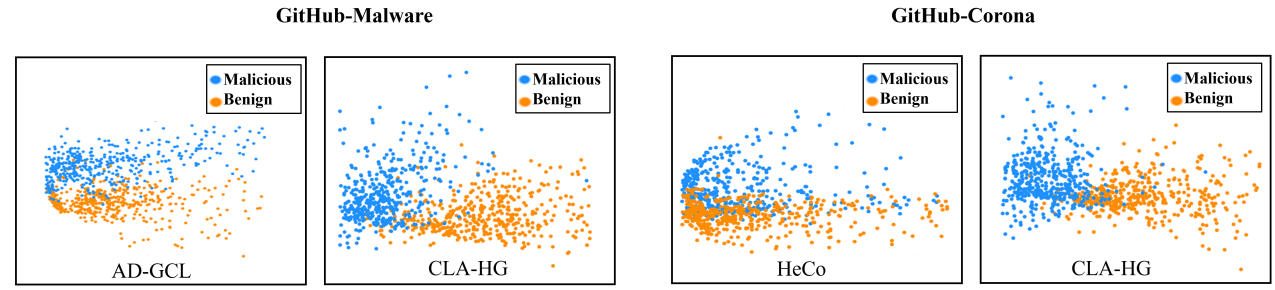

Figure 2: Visualization of repository embedding generated by (AD-GCL, CLA-HG) on GitHub-Malware and (HeCo, CLA-HG) on GitHub-Corona.

structure. In particular, for HAN, we implement HAN by two defined meta-paths (i.e., P_1 and P_2) to learn the attention-based node embedding. Similar to G1, we feed the learned repository embedding to a 2-layer DNN classifier to detect malicious repositories.

Malicious repository detection methods. We reproduce three existing methods that are designed to detect malicious repositories, i.e., GitCyber [59], SourceFinder [32], and Meta-AHIN [29]. Based on the setting of GitCyber, we build a DNN classifier guided by two defined meta-paths on HG to detect malicious repositories. Similar to the setting of SourceFinder [32], we utilize the pre-trained model word2vec [25] to get the embedding for each repository and feed it to a Multinomial Naive Bayes [16] for malicious repository detection. In addition, as the GitHub-Corona dataset is provided by [29], we closely follow the settings in [29] to learn repository representations via a meta-learning framework and further predict the malicious score of repositories.

Graph contrastive learning models. We reproduce GCC [31], GraphCL [52], and HeCo [43] based on their source code. For GCC, we implement GIN as the graph encoder and adopt an end-to-end strategy to build the subgraph dictionary (with size 1023) in contrastive learning. For GraphCL, we adopt edges perturbation and edges masking as graph augmentation methods and GCN as graph encoder to conduct contrastive learning. For HeCo, we leverage two meta-paths (i.e., P_1 and P_2) to closely follow the setting and conduct node-view and meta-path view contrastive learning. We also reproduce two works about graph contrastive learning with adversarial augmentations, i.e., GASSL [50] and AD-GCL [39]. For GASSL, we automatically generate challenging views by adding perturbations

to our graph features and adversarially train the graph encoder. We finally reproduce AD-GCL by leveraging the adversarial attacks as the graph augmentation method to generate contrastive pairs for graph contrastive learning.

5.3 Experimental Setup

To evaluate the performance of CLA-HG and baseline methods, we adopt two widely-used metrics: accuracy showing the percent of repositories being classified correctly (**ACC**) and recall score showing the percent of malicious repositories being detected successfully (**REC**). We conduct 10-fold cross-validation for all models to get the averaged results. Experiments are conducted under the environment of the Ubuntu 16.04 OS, plus Intel i9-9900k CPU, GeForce GTX 2080 Ti Graphics Cards, and 64 GB of RAM. We employ a 2-layer GCN with weight decay 1e-5 as encoder and the dimension of repository embedding generated by encoder is 200. We use Adam [17] optimizer with learning rate 0.001. For dual-stream graph contrastive learning, we sample 2-hop ($m = 2$) subgraphs and the fixed-size numbers of neighbor is set as 10 for the first hop and 5 for the second hop. The temperature parameter τ is set as 0.5. The adversarial hyper-parameter λ_{adv} is set as 0.5. During fine-tuning, we set the temperature parameter t in IKD as 10, α in RKD as 0.6, and the trade-off weight λ_{kd} is set as 0.5. Our source code will be available upon publication.

5.4 Performance Comparison

Table 2 shows the performances of all models on two datasets and the best performance are highlighted in bold and the best baseline results are emphasized by underline. All results listed in the table are the average results of ten runs. The percentage number shows the ratio of training data used for model training while the rest data is used for model validation (10%) and testing. According to the table, we can conclude that: (i) By comparison with G1, G2, and G3, we can find that relations among entities can improve the model performance of repository classification. Merely considering code-based content is not supportive enough to detect malicious repository. In addition, we can also get the conclusion when comparing SourceFiner with GitCyber and Meta-HG in G3. Both GitCyber and Meta-AHIN have better performance than that of SourceFiner as they consider both the content information and relationships of repositories. (ii) By comparison with G2 and G4, we see that unlabeled data contributes greatly when we exploit unlabeled data to train an encoder for graph representation learning, showing the effectiveness of unlabeled data for repository representation learning. (iii) GASSL and AD-GCL, leveraging adversarial attacks as the graph augmentation method to generate contrastive pairs, has better performance than other contrastive learning models in G4 (GCC and GraphCL). This shows that adversarial attacks can generate more challenging contrastive pairs to boost the performance of contrastive learning. (iv) When fine-tuning the pre-trained encoder with different knowledge distillation methods, relational knowledge distillation extracts more useful unlabeled information than individual knowledge distillation. (v) By comparison with all baseline models, CLA-HG is proved to have the best performance, demonstrating the effectiveness of our model design for malicious repository detection.

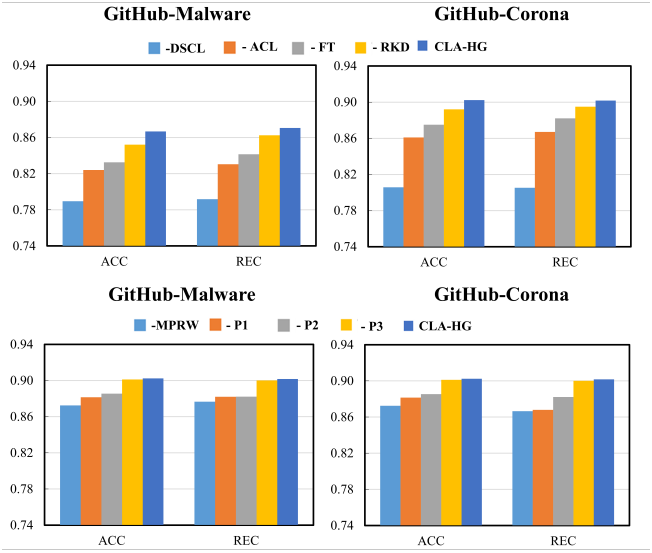

Figure 3: Two sets of ablation studies on two datasets.

5.5 Ablation Study

To further show the effectiveness of all components in CLA-HG and the success of hybrid strategy for heterogeneous subgraph sampling, we conduct two sets of ablation experiments in Figure 3. First, we conduct ablation experiments to analyze the contribution of each component in CLA-HG (i.e., dual-stream contrastive learning (DSCL), adversarial contrastive learning (ACL), fine-tuning(FT), and relational knowledge distillation (RKD)) by removing it separately. We remove DSCL from CLA-HG, which means we directly employ GCN to learn the node embedding in HG and further feed the repository embedding to 2-layer MLP for repository classification. We can conclude that DSCL has the largest contribution to CLA-HG as the performance drops significantly on both datasets. In addition, we remove ACL from CLA-HG and leverage standard heterogeneous subgraphs as contrastive samples to conduct contrastive learning. We can find that the performance also decreases obviously. Afterwards, we remove fine-tuning from CLA-HG, which means we directly apply the repository embedding generated by the pre-trained encoder to a two-layer MLP. It is easy to find that transferring the weights from the pre-trained encoder to the downstream classification model also has a contribution to CLA-HG. Moreover, we remove RKD from CLA-HG and we see that the performance decreases slightly on both datasets, showing the effectiveness of relational knowledge distillation in enhancing the model.

We also conduct extensive ablation experiments to analyze the contribution of each meta-path (i.e., P_1, P_2, P_3) and validate the effectiveness of our hybrid neighbor sampling strategy in heterogeneous subgraph sampling. We first remove the meta-path guided random walk strategy (MPRW), which means we uniformly sample neighbors to generate heterogeneous subgraphs. The performance drops obviously on both datasets, demonstrating the importance of MPRW strategy in subgraph sampling. We then remove each meta-path independently and we find that the performance of model without P_1 or P_2 decreases but the performance of model without P_3 does not change obviously, showing the effectiveness of P_1

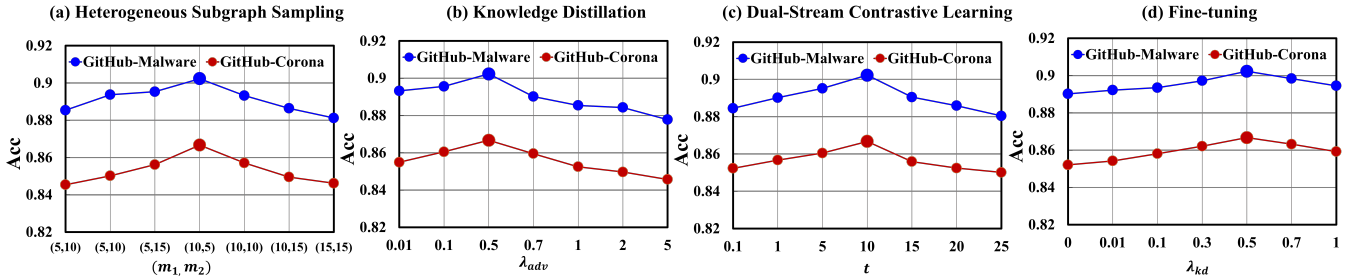

Figure 4: Hyper-parameters sensitivity analysis on two datasets.

and P_2. Hence, we select P_1 and P_2 to guide random walk when sampling heterogeneous subgraphs.

5.6 Comparison with Commercial Products

To further demonstrate the effectiveness of CLA-HG, we compare it with four popular commercial anti-malware products (i.e., AVG, Kaspersky, Comodo, and ESET) and LGTM (provided by GitHub) on two datasets in Table 3. The Version shows the information of the product we used for evaluation. Note that different versions may have different performances due to version biases. We find that CLA-HG achieves at least 2% improvement in accuracy and recall by comparing it with other four products on GitHub-Malware and GitHub-Corona. Among those commercial products, AVG has the best performance on two datasets but it still fails to win CLA-HG. Besides, by comparison with LGTM provided by GitHub, CLA-HG achieves over 20% accuracy and recall improvement on both datasets. Therefore, Table 3 again demonstrates the strong applicabilities of CLA-HG for malicious repository detection in GitHub.

Table 3: Comparison with commercial products.

Method- Version	GitHub-Malware		GitHub-Corona	
	ACC	REC	ACC	REC
CLA-HG	**0.9023**	**0.9017**	**0.8667**	**0.8705**
AVG (19.8.3108)	0.8839	0.8690	0.8478	0.8441
Kaspersky (15.0.1.13)	0.8735	0.8583	0.8428	0.8263
Comodo (32668)	0.8670	0.8502	0.8446	0.8220
ESET (15.0.18)	0.8658	0.8478	0.8415	0.8216
LGTM (GitHub)	0.6913	0.6349	0.6645	0.6198

5.7 Embedding Visualization

In order to show a more intuitive performance comparison, we visualize the embedding of malicious and benign repositories generated by CLA-HG and the best baseline models on GitHub-Malware (AD-GCL) and GitHub-Corona (HeCo) in Figure 2. The blue points refer to malicious repositories and the orange points represent benign repositories. We can find that CLA-HG generates more distinct boundaries and a smaller overlapping area between malicious and benign repositories on GitHub-Malware. In addition, although HeCo shows excellent performance on GitHub-Corona, we find that the overlapping area between malicious and benign repositories is still bigger than CLA-HG. Hence, Figure 2 further demonstrates the superiority of CLA-HG in detecting malicious repositories.

5.8 Hyper-parameters Sensitivity

To explore the hyper-parameters sensitivity in this paper, we conduct four analysis experiments w.r.t. the number of nodes in each hop on heterogeneous subgraph sampling, λ_{adv} on Dual-stream contrastive learning, t on individual knowledge distillation, and λ_{kd} on fine-tuning. Specifically, in Figure 4.(a), we vary the number of neighbors in the first hop and second hop (denoted as m_1, m_2) in the range of $\{5, 10, 15\}$ to generate heterogeneous subgraphs. We can find that the optimal number of neighbors for the first hop and second hop are m_1=10 and m_2=5 respectively. Besides, in Figure 4.(b), we vary the value of λ_{adv} in dual-stream contrastive learning in the range of $\{0.01, 0.1, 0.5, 0.7, 1.0, 2.0, 5.0\}$. We can see that the model performance increases with the increment of λ_{adv} and the optimal value is 0.5, while the performance decreases when λ_{adv} goes beyond the optimal value. In addition, in Figure 4.(c), we vary the value of t in Eq. 10 in the range of $\{0.1, 1, 5, 10, 15, 20, 25\}$ to soften the peaky softmax function on the logits of node embedding in individual knowledge distillation and we find the optimal value of temperature index t is 10. Moreover, in Figure 4.(d), we vary the value of trade-off weight λ_{kd} in Eq. 13 during fine-tuning. By comparing the performance of 0 and 0.5 (optimal value), we can again demonstrate the effectiveness of knowledge distillation in enhancing the performance of the fine-tuned model.

6 CONCLUSION

In this work, we create one new dataset (GitHub-Malware) and develop a novel framework called CLA-HG to solve the imminent problem of malicious repository detection. Specifically, we first build a HG to describe the relations and content information of repository data. Then we design a dual-stream graph contrastive learning framework that distinguishes adversarial subgraph pairs and standard subgraph pairs to pre-train an expressive encoder. Afterwards, we transfer the pre-trained encoder to the downstream repository classification model and further augmented by a knowledge distillation module. The empirical results demonstrate the superior performance of CLA-HG by comparison with baseline methods and commercial anti-malware products.

ACKNOWLEDGMENTS

This work is partially supported by the NSF under grants IIS-2209814, IIS-2203262, IIS-2214376, IIS-2217239, OAC-2218762, CNS-2203261, CNS-2122631, CMMI-2146076, and the NIJ 2018-75-CX-0032. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.

REFERENCES

[1] 2017. virustotal: R Client for the VirusTotal API. https://cran.r-project.org/web/packages/virustotal/index.html.

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *ICML*.

[4] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. 2020. Big Self-Supervised Models are Strong Semi-Supervised Learners. In *NeurIPS*.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*.

[8] Qingyuan Gong, Jiayun Zhang, Yang Chen, Qi Li, Yu Xiao, Xin Wang, and Pan Hui. 2019. Detecting malicious accounts in online developer communities using deep learning. In *CIKM*.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

[10] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).

[11] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied logistic regression*. John Wiley & Sons.

[12] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2020. Strategies for pre-training graph neural networks. In *ICLR*.

[13] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. GPT-GNN: Generative Pre-Training of Graph Neural Networks. In *KDD*.

[14] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In *KDD*.

[15] Nikola Jovanović, Zhao Meng, Lukas Faber, and Roger Wattenhofer. 2021. Towards robust graph contrastive learning. *arXiv preprint arXiv:2102.13085* (2021).

[16] Ashraf M Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. 2004. Multinomial naive bayes for text categorization revisited. In *AJCAI*.

[17] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

[18] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[19] William La Cholter, Matthew Elder, and Antonius Stalick. 2021. Windows Malware Binaries in C/C++ GitHub Repositories: Prevalence and Lessons Learned. In *ICISSP*.

[20] Zemin Liu, Vincent W Zheng, Zhou Zhao, Zhao Li, Hongxia Yang, Minghui Wu, and Jing Ying. 2018. Interactive paths embedding for semantic proximity search on heterogeneous graphs. In *KDD*.

[21] Zemin Liu, Vincent W Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. 2017. Semantic proximity search on heterogeneous graph by proximity embedding. In *AAAI*.

[22] Yuanfu Lu, Xunqiang Jiang, Yuan Fang, and Chuan Shi. 2021. Learning to pre-train graph neural networks. In *AAAI*.

[23] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *ICLR*.

[24] Alessio Micheli. 2009. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks* (2009).

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[26] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. 2019. Relational Knowledge Distillation. In *CVPR*.

[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*.

[28] Yiyue Qian, Yiming Zhang, Qianlong Wen, Yanfang Ye, and Chuxu Zhang. 2022. Rep2Vec: Repository Embedding via Heterogeneous Graph Adversarial Contrastive Learning. In *KDD*.

[29] Yiyue Qian, Yiming Zhang, Yanfang Ye, and Chuxu Zhang. 2021. Adapting Meta Knowledge with Heterogeneous Information Network for COVID-19 Themed Malicious Repository Detection. In *IJCAI*.

[30] Yiyue Qian, Yiming Zhang, Yanfang Ye, Chuxu Zhang, et al. 2021. Distilling Meta Knowledge on Heterogeneous Graph for Illicit Drug Trafficker Detection on Social Media. In *NeurIPS*.

[31] 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 *KDD*.

[32] Md Omar Faruk Rokon, Risul Islam, Ahmad Darki, Evangelos E Papalexakis, and Michalis Faloutsos. 2020. Sourcefinder Finding malware source-code from publicly available repositories in github. In *RAID*.

[33] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*.

[34] Semmle. 2019. CodeQL for research. https://securitylab.github.com/tools/codeql.

[35] Semmle. 2019. LGTM. https://github.com/marketplace/lgtm.

[36] Donald F Specht et al. 1991. A general regression neural network. *IEEE transactions on neural networks* (1991).

[37] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2020. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *ICLR*.

[38] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. In *VLDB*.

[39] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. 2021. Adversarial graph augmentation to improve graph contrastive learning. In *NeurIPS*.

[40] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[41] Lin Wang and Kuk-Jin Yoon. 2021. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[42] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *WWW*.

[43] Xiao Wang, Nian Liu, Hui Han, and Chuan Shi. 2021. Self-supervised Heterogeneous Graph Neural Network with Co-contrastive Learning. In *KDD*.

[44] Wikipedia. 2022. GitHub Introduction. https://en.wikipedia.org/wiki/GitHub.

[45] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[46] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Haifeng Chen, and Xiang Zhang. 2021. Infogcl: Information-aware graph contrastive learning. In *NeurIPS*.

[47] Guodong Xu, Ziwei Liu, Xiaoxiao Li, and Chen Change Loy. 2020. Knowledge distillation meets self-supervision. In *ECCV*.

[48] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *ICLR*.

[49] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. 2020. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[50] Longqi Yang, Liangliang Zhang, and Wenjing Yang. 2021. Graph Adversarial Self-Supervised Learning. In *NeurIPS*.

[51] Yanfang Ye, Yujie Fan, Shifu Hou, Yiming Zhang, Yiyue Qian, Shiyu Sun, Qian Peng, Mingxuan Ju, Wei Song, and Kenneth Loparo. 2020. Community mitigation: A data-driven system for covid-19 risk assessment in a hierarchical manner. In *CIKM*.

[52] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. In *NeurIPS*.

[53] Lu Yu, Shichao Pei, Lizhong Ding, Jun Zhou, Longfei Li, Chuxu Zhang, and Xiangliang Zhang. 2022. SAIL: Self-Augmented Graph Contrastive Learning. In *AAAI*.

[54] Yujia Zhai, Wei Song, Xianjun Liu, Lizhen Liu, and Xinlei Zhao. 2018. A chi-square statistics based feature selection method in text classification. In *ICSESS*.

[55] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *KDD*.

[56] Chuxu Zhang, Ananthram Swami, and Nitesh V Chawla. 2019. Shne: Representation learning for semantic-associated heterogeneous networks. In *WSDM*.

[57] Chuxu Zhang, Lu Yu, Xiangliang Zhang, and Nitesh V Chawla. 2018. Task-guided and semantic-aware ranking for academic author-paper correlation inference. IJCAI.

[58] Xiang Zhang and Marinka Zitnik. 2020. Gnnguard: Defending graph neural networks against adversarial attacks. In *NeurIPS*.

[59] Yiming Zhang, Yujie Fan, Shifu Hou, Yanfang Ye, Xusheng Xiao, Pan Li, Chuan Shi, Liang Zhao, and Shouhuai Xu. 2020. Cyber-guided Deep Neural Network for Malicious Repository Detection in GitHub. In *ICKG*.

[60] Yiming Zhang, Yiyue Qian, Yujie Fan, Yanfang Ye, Xin Li, Qi Xiong, and Fudong Shao. 2020. dstyle-gan: Generative adversarial network based on writing and photography styles for drug identification in darknet markets. In *ACSAC*.

[61] Yiming Zhang, Yiyue Qian, Yanfang Ye, and Chuxu Zhang. 2022. Adapting Distilled Knowledge for Few-shot Relation Reasoning over Knowledge Graphs. In *SDM*.

[62] Jianan Zhao, Qianlong Wen, Shiyu Sun, Yanfang Ye, and Chuxu Zhang. 2021. Multi-view Self-supervised Heterogeneous Graph Embedding. In *ECML/PKDD*.