Out-of-sample Node Representation Learning for Heterogeneous Graph in Real-time Android Malware Detection

Yanfang Ye[1,2*], **Shifu Hou**[1,2], **Lingwei Chen**[2], **Jingwei Lei**[3], **Wenqiang Wan**[3], **Jiabin Wang**[3], **Qi Xiong**[3] and **Fudong Shao**[3]

[1]Department of CDS, Case Western Reserve University, OH, USA
[2]Department of CSEE, West Virginia University, WV, USA
[3]Tencent Security Lab, Tencent, Guangdong, China

Abstract

The increasingly sophisticated Android malware calls for new defensive techniques that are capable of protecting mobile users against novel threats. In this paper, we first extract the runtime Application Programming Interface (API) call sequences from Android apps, and then analyze higher-level semantic relations within the ecosystem to comprehensively characterize the apps. To model different types of entities (i.e., *app*, *API*, *device*, *signature*, *affiliation*) and rich relations among them, we present a structured heterogeneous graph (HG) for modeling. To efficiently classify nodes (e.g., apps) in the constructed HG, we propose the *HG-Learning* method to first obtain in-sample node embeddings and then learn representations of out-of-sample nodes without rerunning/adjusting HG embeddings at the first attempt. We later design a deep neural network classifier taking the learned HG representations as inputs for real-time Android malware detection. Comprehensive experiments on large-scale and real sample collections from Tencent Security Lab are performed. Promising results demonstrate that our developed system *AiDroid* which integrates our proposed method outperforms others in real-time Android malware detection.

1 Introduction

Due to the mobility and ever expanding capabilities, smart phones have become increasingly ubiquitous in people's everyday life performing tasks such as social networking and online banking. Android, as an open source and customizable operating system (OS) for smart phones, is currently dominating the smart phone market by 75.16% [Statcounter, 2018]. However, due to its large market share and open source ecosystem of development, Android attracts not only the developers for producing legitimate Android applications (apps), but also attackers to disseminate malware (***mal***icious soft***ware***) that deliberately fulfills the harmful intent to the smart phone users (e.g., stealing user credentials, pushing unwanted apps or advertisements). Driven by the considerable economic profits, there has been explosive growth of Android malware which posed serious threats to the smart phone users; it's reported that there have been $4,687,008$ newly generated Android malware that infected more than 61 million smart phones in the first half of 2018 [TencentSecurity, 2018]. To evade the detection by mobile security products, Android malware has turned to be increasingly sophisticated. For example, as shown in Figure 1, the "TigerEyeing" trojan is a new kind of Command and Control (C&C) malware that pretends to be a legitimate app (e.g., mobile game, system tool) and only executes to perform the profitable tasks on-demand. The explosive growth and increasing sophistication of Android malware call for new defensive techniques that are capable of protecting mobile users against novel threats.

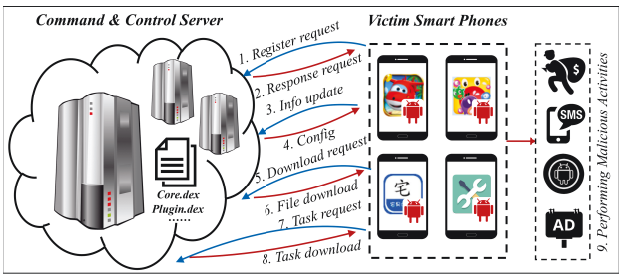

Figure 1: Increasingly sophisticated Android malware.

To combat the evolving Android malware attacks, in this paper, we first extract the Application Programming Interface (API) call sequences from runtime executions of Android apps in users' smart phones to capture their behaviors. We further analyze higher-level semantic relationships within the ecosystem, such as whether two apps have similar behaviors, whether they co-exist in the same smart phone that can be identified by its unique International Mobile Equipment Identity (IMEI) number, whether they are signed by the same developer or produced by the same company (i.e., affiliation), etc. To depict such complex relationships, we present a structured heterogeneous graph (HG) [Sun and Han, 2012] for modeling. To efficiently classify nodes (e.g., apps) in the constructed HG, HG embedding methods [Fu *et al.*, 2017; Dong *et al.*, 2017; Fan *et al.*, 2018] have been proposed; unfortunately, most of these existing methods are primarily designed for static HGs, where all nodes are known before

*Corresponding author: yanfang.ye@mail.wvu.edu

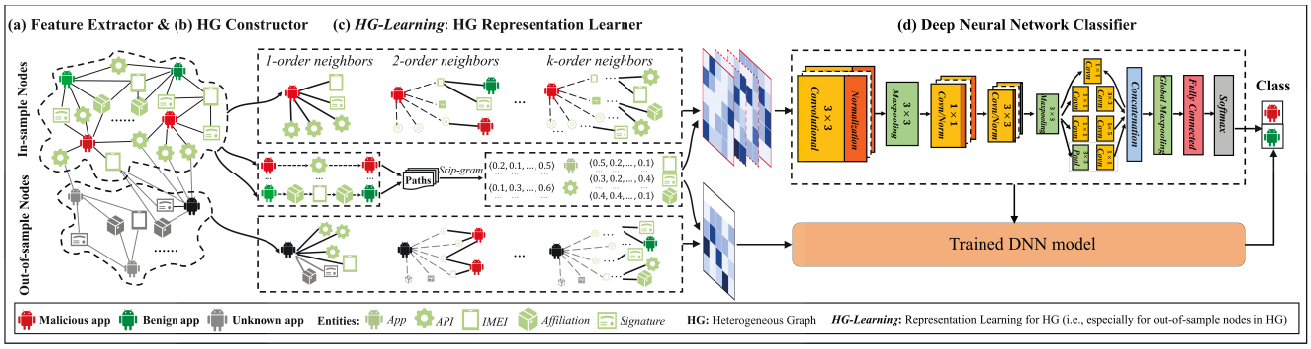

Figure 2: Overview of the developed system *AiDroid* for real-time Android malware detection.

learning. Systems such as HinDroid [Hou *et al.*, 2017] and Scorpion [Fan *et al.*, 2018] have demonstrated the success of HG based models in malware detection; however, they are also developed based on static HGs and thus fail to deal with the problem of out-of-sample node representation learning. As our application requires real-time prediction of new coming nodes (i.e., unknown apps), it is infeasible to rerun HG embeddings whenever new nodes arrive, especially considering the fact that rerunning HG embeddings also results in the need of retraining the downstream classifier. How to efficiently learn the representations of out-of-sample nodes in HG remains largely unanswered. To solve this problem, we propose the *HG-Learning* method to first obtain in-sample node embeddings and then learn representations of out-of-sample nodes without rerunning/adjusting HG embeddings at the first attempt. Afterwards, we design a deep neural network (DNN) classifier leveraging the advantages of convolutional neural networks (CNNs) and Inception for Android malware detection. We develop a system *AiDroid* (shown in Figure 2) integrating our proposed method for real-time Android malware detection, which has major merits of:

- Besides runtime behaviors, we further analyze the complex relations within the ecosystem to characterize Android apps. Then, we present a structured HG for modeling. This provides a comprehensive solution that is more resilient against Android malware's evasion tactics.
- To solve the node (i.e., app) classification problem in HG, we propose the *HG-Learning* method to efficiently learn representations of out-of-sample nodes in HG using in-sample node embeddings while without rerunning/adjusting them for the first time, which makes the downstream classifier feasible for classifying new arriving nodes without retraining. Though it's proposed for real-time Android malware detection, the *HG-Learning* method is a general framework to learn desirable node representations in HG (i.e., especially for out-of-sample nodes) and thus it can be further applied to various speed-sensitive dynamic graph/network mining tasks.
- We provide comprehensive experimental studies based on the large-scale and real sample collections from Tencent Security Lab, which demonstrate the performance of *AiDroid*. It has been incorporated into Tencent Mobile Security product that protects millions of users.

2 Proposed Method

In this section, we introduce the proposed method which is integrated in *AiDroid* in detail.

2.1 Feature Extraction

Dynamic Behavior Extraction. API calls are used by Android apps in order to access Android OS functionality and system resources. Therefore, we extract the sequences of API calls in the application framework from runtime executions of Android apps to capture their behaviors. For example, a sequence of API calls (*StartActivity*, *checkConnect*, *getPhoneInfo*, *receiveMsg*, *sendMsg*, *finishActivity*) extracted from the previous mentioned "TigerEyeing" trojan represents its typical behaviors of connecting to the C&C server in order to fetch the configuration information; while another sequence of its extracted API calls (*startActivity*, *checkConnect*, *sendSMS*, *finishActivity*) denotes its intention of sending SMS messages without user's concern.

Relation-based Feature Extraction. Besides the above extracted API call sequences that can be used to represent an app's behaviors, to detect the increasingly sophisticated Android malware, we further consider various kinds of relations: i) *R1*: the *app-invoke-API* relation describes whether an app invokes an API call in runtime execution; ii) *R2*: the *app-exist-IMEI* relation describes whether an app exists (i.e., is installed) in a smart phone (i.e., IMEI); iii) *R3*: the *app-certify-signature* relation means an app is certified by a signature, denoting that it's signed by a developer; iv) *R4*: package name (a.k.a. Google Play ID) is an unique identifier for an Android app; since companies conventionally use their reserved domain name to begin their package names (e.g., "com.tencent.mobileqq"), we extract the domain name from the package name and build *app-associate-affiliation* to indicate the relation between an app (e.g., "mobileqq") and its affiliation (e.g., "tencent.com"); v) *R5*: the *IMEI-have-signature* relation represents if a smart phone has a set of apps signed by a particular developer; vi) *R6*: the *IMEI-possess-affiliation* relation denotes if a smart phone installs a set of apps associated with a specific affiliation.

2.2 Heterogeneous Graph Construction

In order to depict apps using the extracted features, we introduce heterogeneous graph (HG) to model them in a proper

way so that different kinds of entities and relations can be better handled. We first present the related concepts as follows.

Definition 1 *A **heterogeneous graph (HG)** [Sun and Han, 2012] is defined as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with an entity type mapping ϕ: $\mathcal{V} \rightarrow \mathcal{A}$ and a relation type mapping ψ: $\mathcal{E} \rightarrow \mathcal{R}$, where $\mathcal{V}$ denotes the entity set and $\mathcal{E}$ is the relation set, $\mathcal{A}$ denotes the entity type set and $\mathcal{R}$ is the relation type set, and the number of entity types $|\mathcal{A}| > 1$ or the number of relation types $|\mathcal{R}| > 1$. The **network schema** [Sun and Han, 2012] for a HG $\mathcal{G}$, denoted as $\mathcal{T}_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$, is a graph with nodes as entity types from $\mathcal{A}$ and edges as relation types from $\mathcal{R}$.*

Based on the definition above, the network schema in our application with five entity types and six types of relations among them is shown in Figure 3, which enables the apps to be represented in a comprehensive way that utilizes their semantic and structural information.

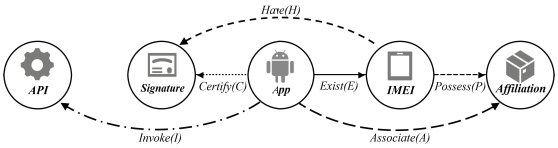

Figure 3: Network schema for HG in our application.

To formulate the relatedness among entities in HG, the concept of meta-path has been proposed [Sun and Han, 2012]: a ***meta-path*** $\mathcal{P}$ is a path defined on the graph of network schema $\mathcal{T}_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$, and is denoted in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} ... \xrightarrow{R_L} A_{L+1}$, which defines a composite relation $R = R_1 \cdot R_2 \cdot \ldots \cdot R_L$ between types A_1 and A_{L+1}, where $\cdot$ denotes relation composition operator, and L is length of $\mathcal{P}$. In our application, based on the HG schema shown in Figure 3, incorporated the domain knowledge from anti-malware experts, we design six meaningful meta-paths to characterize the relatedness over apps at different views (i.e., *PID1*-*PID6* shown in Figure 4). For example, *PID1* depicts that two apps are related if they both invoke the same API (e.g., two malicious mobile video players both invoke the API of "*requestAudioFocus*"); while *PID4* depicts that two apps are connencted if they are developed by the same developer.

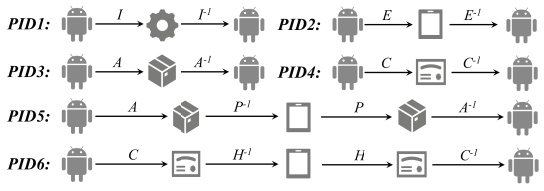

Figure 4: Meta-paths built for Android malware detection.

2.3 *HG-Learning*: Representation Learning of Out-of-Sample Nodes in HG

Traditional network representation learning mainly focuses on factorizing the matrix (e.g., adjacency matrix) of network to generate latent-dimension features for the nodes. However, the computational cost of decomposing a large-scale matrix is usually very expensive, and also suffers from its statistical performance drawback [Grover and Leskovec, 2016]. Since Android malware detection is a speed sensitive application and requires cost-effective solutions, scalable representation learning method for HG, especially for out-of-sample nodes, is in need. To address these challenges, we first formalize HG representation learning.

Definition 2 ***HG Representation Learning*** *[Fu* et al.*, 2017; Dong* et al.*, 2017]. Given a HG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the representation learning task is to learn a function $f : \mathcal{V} \rightarrow \mathbb{R}^d$ that maps each node $v \in \mathcal{V}$ to a vector in a d-dimensional space $\mathbb{R}^d$, $d \ll |\mathcal{V}|$ that are capable to preserve the structural and semantic relations among them.*

To solve the problem of HG representation learning, due to the heterogeneous property of HG (i.e., graph consisting of multi-typed entities and relations), it is difficult to directly apply the conventional homogeneous graph embedding techniques [Perozzi *et al.*, 2014; Tang *et al.*, 2015; Grover and Leskovec, 2016] to learn the latent representations for HG. To address this issue, HG embedding methods [Fu *et al.*, 2017; Dong *et al.*, 2017; Fan *et al.*, 2018] have been proposed, which are capable to preserve the semantic and structural correlations between different types of nodes. However, most of these existing models are primarily designed for static HGs, where all nodes are known before learning. In our application (i.e., real-time Android malware detection), it is infeasible to rerun HG embeddings whenever new nodes arrive, especially considering the fact that rerunning HG embeddings also results in the need of retraining the downstream classifier. How to efficiently learn the representations of out-of-sample nodes in HG, i.e., nodes that arrive after the embedding process, remains largely unanswered. To solve this problem, we propose HG in-sample node embedding model (i.e., *HGiNE*) to learn in-sample node embeddings that is able to preserve the heterogeneous property of HG; and then we devise *HG2Img* model to learn out-of-sample node representations and also enrich in-sample node representations using learned in-sample node embeddings without rerunning/adjusting HG embeddings.

In-sample node embedding (HGiNE). We first propose a random walk strategy guided by different meta-paths to map the word-context concept into a HG; then exploit skip-gram to learn effective in-sample node representations for a HG.

Given a source node v_j in a homogeneous graph, the traditional random walk is a stochastic process with random variables $v_j^1, v_j^2, ..., v_j^k$ such that v_j^{k+1} is a node chosen at random from the neighbors of node v_k. The transition probability $p(v_j^{i+1}|v_j^i)$ at step i is the normalized probability distributed over the neighbors of v_j^i by ignoring their node types. However, this mechanism is unable to capture the semantic and structural correlations among different types of nodes in a HG. Here, we show how we use different built meta-paths to guide the random walker in a HG to generate the paths of multiple types of nodes. Given a HG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with schema $\mathcal{T}_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$, and a set of different meta-paths $\mathcal{S} = \{\mathcal{P}_j\}_{j=1}^n$, each of which is in the form of $A_1 \rightarrow ...A_t \rightarrow A_{t+1}... \rightarrow A_l$,

we put a random walker to traverse the HG. The random walker will first randomly choose a meta-path $\mathcal{P}_k$ from $\mathcal{S}$ and the transition probabilities at step i are defined as:

$$p(v^{i+1}|v^i_{A_t}, \mathcal{S}) = \begin{cases} \frac{\lambda}{|\mathcal{S}|}\frac{1}{|N_{A_{t+1}}(v^i_{A_t})|} \\ \quad \text{if } (v^{i+1}, v^i_{A_t}) \in \mathcal{E}, \phi(v^i_{A_t}) = A_{app}, \phi(v^{i+1}) = A_{t+1} \\ \frac{1}{|N_{A_{t+1}}(v^i_{A_t})|} \\ \quad \text{if } (v^{i+1}, v^i_{A_t}) \in \mathcal{E}, \phi(v^i_{A_t}) \neq A_{app}, \\ \quad \phi(v^{i+1}) = A_{t+1}, (A_t, A_{t+1}) \in \mathcal{P}_k \\ 0 \quad \text{otherwise}, \end{cases} \tag{1}$$

where $N_{A_{t+1}}(v^i_{A_t})$ denotes the A_{t+1} type of neighborhood of node $v^i_{A_t}$, A_{app} is entity type of *app*, and λ is the number of meta-paths starting with $A_{app} \rightarrow A_{t+1}$ in the given meta-path set $\mathcal{S}$. The walk paths generated by the above strategy are able to preserve both the semantic and structural relations between different types of nodes in the HG.

After mapping the word-context concept in a text corpus into a HG via the above proposed meta-path guided random walk strategy (i.e., a sentence in the corpus corresponds to a sampled path and a word corresponds to a node), skip-gram [Mikolov *et al.*, 2013a; Perozzi *et al.*, 2014] is then applied on the paths to minimize the loss of observing a node's neighborhood (within a window w) conditioned on its current representation. The objective function of skip-gram is:

$$\arg\min_{X} \sum_{-w \leq k \leq w, j \neq k} -\log p(v_{j+k}|X(v_j)), \tag{2}$$

where $X(v_j)$ is the current representation vector of v_j, and $p(v_{j+k}|X(v_j))$ is defined using the softmax function:

$$p(v_{j+k}|X(v_j)) = \frac{\exp(X(v_{j+k}) \cdot X(v_j))}{\sum_{q=1}^{|\mathcal{V}|} \exp(X(v_q) \cdot X(v_j))}. \tag{3}$$

Due to its efficiency, we first apply hierarchical softmax technique [Mikolov *et al.*, 2013b] to solve Eq. 3, and then employ the stochastic gradient descent to train the skip-gram.

Out-of-sample node representation learning (HG2Img). Can we use in-sample node embeddings learned by the above proposed HGiNE to efficiently learn representations of out-of-sample nodes in HG and also enrich in-sample node representations? To answer this question, we first present the concept of k-order neighbors in HG as following.

Definition 3 ***k-order Neighbors in HG.*** *Given a HG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, let 1-order neighbors of a node $v_i \in \mathcal{V}$ be $S^{(1)}(v_i)$ so that $S^{(1)}(v_i) = \{v_j | (v_i, v_j) \in \mathcal{E}\}$; then, k-order neighbors $S^{(k)}(v_i)$ of a node v_i ($k > 1$) can be denoted as $S^{(k)}(v_i) = \{S^{(1)}(v_z) \setminus S^{(k-2)}(v_i), v_z \in S^{(k-1)}(v_i)\}$.*

Based on the above definition, each node (i.e., in-sample or out-of-sample) in HG can be represented by the following representation matrix, which leverages not only the embedding of itself but also the information of its k-order neighbors:

$$\mathbf{X}(v_i) = [X(v_i), X(S^{(1)}(v_i)), ..., X(S^{(k)}(v_i))], \tag{4}$$

where $\mathbf{X} \in \mathbb{R}^{t \times d}$, each of which denotes d-dimensional node embedding. In our application, as illustrated in Figure 5, for each k, we exploit breadth-first search (BFS) method to find t_k neighbors in the order of type of app, signature, package, IMEI and API, and $t = 1 + \sum_1^k t_k$. Empirically we find $k = 2$ and $t = d$ perform the best, and apply them to our problem. Note that we zero-pad the representation matrix in the corresponding rows when the node embeddings cannot be found (i.e., out-of-sample nodes) or $t < d$.

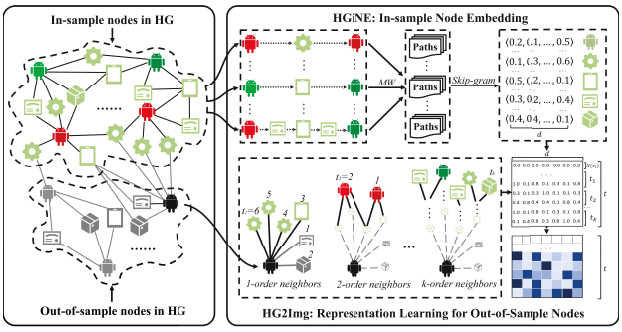

Figure 5: *HG-Learning*: node representation learning in HG.

Resting on the in-sample node embeddings which can be learned offline using the proposed HGiNE, when a new node (e.g., a testing app) arrives, it only takes $O(t \times d)$ time to obtain its representation using the proposed HG2Img; furthermore, the representation learning for the new arriving node doesn't require rerunning HG embeddings, which makes the downstream classifier workable for classifying the new arriving node without retraining. Note that, to enrich the representation of each in-sample-node, after exploiting HGiNE to obtain its embedding, we further apply HG2Img to generate a $t \times d$ matrix $\mathbf{X}$ as its final representation.

2.4 Deep Neural Network Classifier

CNNs [LeCun *et al.*, 2015] have achieved great success in learning salient features for classification tasks; while the crafty architecture of Inception [Szegedy *et al.*, 2015] has shown high performance and low cost under strict constraints on memory and computational budget. Therefore, taking the generated representations of Android apps from previous section as inputs, we devise our deep learning framework leveraging the advantages of CNNs [Huang *et al.*, 2018] and Inception for real-time Android malware detection, which is illustrated in Figure 2.(d). In our designed DNN: (1) for *training*, each in-sample node with type of app in HG represented by a $t \times d$ matrix $\mathbf{X}$ will be fed as an input to the multilayer architecture to learn the higher level concept and train the classification model; (2) for *prediction*, each new arriving node (i.e., out-of-sample node with type of app) will first obtain its representation in real-time by using the embeddings of its k-order neighbors (i.e., in-sample-nodes) and then it will be predicted by the trained model as either benign or malicious. Algorithm 1 illustrates the implementation of our developed system *AiDroid* which integrates the above proposed method.

3 Experimental Results and Analysis

In this section, we fully evaluate the performance of *AiDroid* for real-time Android malware detection.

Algorithm 1: *AiDroid* for malware detection.

Input: HG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with schema $\mathcal{T}_\mathcal{G} = (\mathcal{A}, \mathcal{R})$, traning data set: D_I (in-sample apps), testing data set: D_O (out-of-sample apps).
Output: $\mathbf{y}$: The labels for the testing data set.

Learn in-sample node embeddings $X(v_i) \in \mathbb{R}^d$ $(i = 1, ..., |\mathcal{V}|)$ using HGiNE;
for $i = 1 \rightarrow |D_I|$ **do**
  Apply HG2Img: for each k, find k-order neighbors $X(S^{(k)}(v_i))$ and then generate the $t \times d$ matrix $\mathbf{X}(v_i) = [X(v_i), X(S^{(1)}(v_i)), ..., X(S^{(k)}(v_i))]$;
end
Train DNN using $\mathbf{X}$s $\in \mathbb{R}^{t \times d}$;
for $j = 1 \rightarrow |D_O|$ **do**
  Use HG2Img to generate $\mathbf{X}(v_j)$;
  Obtain the label y_{v_j} using trained DNN;
end
return $\mathbf{y}$;

3.1 Data Collection

We obtain the large-scale and real sample collections from Tencent Security Lab, which contain 190,696 training apps (i.e., 83,784 benign and 106,912 malicious). After feature extraction, the constructed HG has 286,421 nodes (i.e., 190,696 nodes with type of app, 331 with type of API, 70,187 with type of IMEI, 8,499 with type of signature, and 16,708 with type of affiliation) and 4,170,047 edges including relations of *R1-R6*. The new coming 17,746 unknown apps with their extracted features are used as testing data (to obtain the ground truth, they are further analyzed by the anti-malware experts, where 13,313 are benign and 4,433 are malicious).

3.2 Baseline Methods

We validate the performance of our proposed method in *AiDroid* by comparisons with different groups of baselines.

First, based on the constructed HG described above, we evaluate our proposed *HG-Learning* method by comparisons with following baselines: (1) **In-sample node embedding**. We compare our proposed HGiNE with other graph embedding methods including: i) DeepWalk and LINE: for DeepWalk [Perozzi *et al.*, 2014] and LINE [Tang *et al.*, 2015], we ignore the heterogeneous property of HG and directly feed the HG for embedding; ii) metapath2vec: we use meta-path scheme separately to guide random walks in metapath2vec [Dong *et al.*, 2017]. For HGiNE, we divide the designed meta-paths into three sets (i.e., $\mathcal{S}_1$ = {*PID1*}, $\mathcal{S}_2$ = {*PID3, PID4*}, $\mathcal{S}_3$ = {*PID2, PID5, PID6*}) and use the proposed strategy to guide random walks. The parameter settings used for HGiNE are in line with the baselines, which are empirically set as: vector dimension $d = 64$ (LINE: 64 for each order (1st- and 2nd-order)), walks per node $r = 20$, walk length $l = 50$ and window size $w = 5$. (2) **Out-of-sample node representation learning**. We compare our proposed HG2Img for out-of-sample node representation learning with following baselines: i) LocalAvg: the out-of-sample nodes are represented by averaging embeddings of neighboring in-sample nodes; ii) LabelProp: label propagation for multivariate regression problem can be used to learn the representations of out-of-sample nodes; as it has been demonstrated [Ma *et al.*, 2018] that the vanilla version [Zhu and Ghahramani, 2002] outperforms others, we hence use it as a baseline; iii) Rerunning: for comparisons, we also run a baseline by rerunning all node embeddings when new nodes arrive.

Second, we evaluate different types of features for Android malware detection. Our proposed method is general for HGs. Thus, a natural baseline is to see whether the knowledge we add in should be represented as HG instead of other features. Here we compare two types of features as follows: (1) **Behavioral Sequences (*f-1*)**. We devise three baselines based on the extracted API call sequences of Android apps: i) we build support vector machine (SVM) classifier based on binary (i.e., if an API call is invoked by an app) feature vectors (i.e., Bin+SVM); ii) we exploit Long Short-term Memory (LSTM) [Sutskever *et al.*, 2014] for sequence modeling, based on which SVM classifier is then built (i.e., LSTM+SVM); and iii) we also train our proposed DNN based on the extracted API call sequences for evaluation (i.e., Seq+DNN). (2) **HG-based Features (*f-2*)**. Based on the constructed HG, the proposed *HG-Learning* is applied for representation learning (i.e., both in-sample and out-of-sample nodes), based on which the designed DNN is used to train the classification model for prediction (i.e., *AiDroid*).

3.3 Comparisons and Analysis

Based on the constructed HG, using the in-sample-nodes for training and the new coming nodes (i.e., apps) for testing, the results of different methods are shown in Table 1.

Metric	In-sample Embedding	Out-of-sample Learning			Rerunning
		LocalAvg	LabelProp	**HG2Img**	
ACC	DeepWalk	0.9057	0.9214	0.9506	0.9516
	LINE	0.9111	0.9307	0.9690	0.9602
	metapath2vec	0.9289	0.9448	0.9799	0.9722
	HGiNE	0.9389	0.9533	**0.9908**	0.9843
F1	DeepWalk	0.8267	0.8547	0.9055	0.9076
	LINE	0.8364	0.8705	0.9398	0.9234
	metapath2vec	0.8669	0.8954	0.9606	0.9459
	HGiNE	0.8849	0.9094	**0.9817**	0.9691

Table 1: Comparisons of different methods.

From Table 1, we can observe that different combinations of in-sample node embedding and out-of-sample representation learning show different performances in Android malware detection: (1) For in-sample node embedding methods, our proposed HGiNE outperforms all baselines in terms of accuracy (*ACC*) and *F1*. That is to say, HGiNE learns significantly better node (i.e., app) representation in HG than current state-of-the-art methods. (2) For out-of-sample representation learning, our proposed HG2Img significantly outperforms all baselines (i.e., the detection performance achieve superb 0.9908 *ACC* and 0.9817 *F1*), which even surpasses the rerunning HG embeddings. Obviously, $t \times d$ representation

matrices learned by HG2Img utilizing 1- and 2-order neighbors are more expressive than other embeddings in depicting the apps for the problem of Android malware detection.

The comparisons of different features in Android malware detection are shown in Table 2, from which we can see that: (1) Based on the extracted API call sequences (i.e., *f-1*), LSTM provides significant improvement in sequence modeling while our proposed DNN outperforms others in Android malware detection. (2) Compared by content-based features only, HG-based features (i.e., *f-2*) indeed perform better. The reason behind this is that HG-based features are more expressive to characterize complex relatednesses over apps which consist of relations between apps and their invoked API calls, and higher-level semantics within the ecosystem.

Feature	Method	TP	TN	FP	FN
	Bin+SVM	3,926	11,828	1,485	507
f-1	LSTM+SVM	4,115	12,339	974	318
	Seq+DNN	4,168	12,504	809	265
f-2	***AiDroid***	4,395	13,188	125	38
Feature	Method	Recall	Precision	ACC	F1
	Bin+SVM	0.8856	0.7255	0.8877	0.7976
f-1	LSTM+SVM	0.9282	0.8086	0.9271	0.8642
	Seq+DNN	0.9402	0.8374	0.9394	0.8858
f-2	***AiDroid***	0.9914	0.9723	0.9908	0.9817

Table 2: Comparisons of different types of features.

3.4 Parameter Sensitivity, Scalability and Stability

In this set of experiments, we evaluate the system performance of *AiDroid*. As shown in Figure 6.(a) and (b), we can see that *AiDroid* is not strictly sensitive to the parameters and is able to reach high performance under a cost-effective parameter choice. We then run the experiments using new arriving apps over a long time span (i.e., 10 days) to assess the average detection time and accuracy. Figure 6.(c) and (d) demonstrate *AiDroid* is scalable and stable over a long time span in detecting newly generated Android malware (i.e., prediction time on average: 4.3 ms/app and ACC on average: 0.9891). Figure 6.(e) shows the ROC curve of *AiDroid* based on the data described in Section 4.1 which achieves an impressive 0.9914 true positive rate (*TPR*) at 0.0094 false positive rate (*FPR*). We can conclude that *AiDroid* is indeed feasible in practical use for real-time Android malware detection.

4 Related Work

In recent years, there have been many malware detection systems developed using machine learning techniques [Ye *et al.*, 2010; Ye *et al.*, 2011; Wu *et al.*, 2012; Wu and Hung, 2014; Hou *et al.*, 2016; Chen *et al.*, 2017; Ye *et al.*, 2017; Saracino *et al.*, 2018]. In particular, systems such as HinDroid [Hou *et al.*, 2017] and Scorpion [Fan *et al.*, 2018] have demonstrated the success of HG based models in malware detection; however, they are developed based on static HGs without considering the new arriving nodes after the HG embedding process. To solve the problem of HG representation learning, HIN2vec

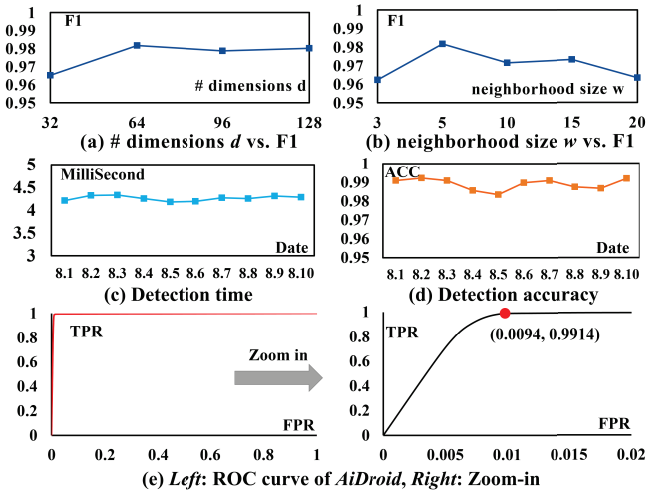

Figure 6: Parameter sensitivity, scalability and stability.

[Fu *et al.*, 2017], metapath2vec [Dong *et al.*, 2017], metagraph2vec [Fan *et al.*, 2018], snippet2vec [Ye *et al.*, 2018], and PME [Chen *et al.*, 2018] have been proposed; however, few of them can deal with out-of-sample nodes. To tackle this problem, though algorithms such as [Chang *et al.*, 2015; Zhao *et al.*, 2018] were proposed to infer embeddings for out-of-sample nodes in HG, they necessitated adjusting in-sample node embeddings and also the downstream classifier retraining. Efficient representation learning for out-of-sample nodes in HG without rerunning/adjusting HG embeddings is in need for the application of real-time Android malware detection. This work is the first attempt to bridge this gap.

5 Conclusion

To combat the evolving Android malware attacks, in this paper, we first extract the API call sequences from runtime executions of Android apps and then analyze higher-level semantic relations within the ecosystem. To depict the complex relations among multi-typed entities, we introduce HG for modeling. To efficiently classify nodes (i.e., apps) in HG, we propose the *HG-Learning* method to first obtain in-sample node embeddings and then learn representations of out-of-sample nodes without rerunning/adjusting HG embeddings for the first time. Afterwards, we design a DNN classifier leveraging the advantages of CNNs and Inception for real-time Android malware detection. Comprehensive experiments on the large-scale and real data collections from Tencent Security Lab demonstrate that our developed system *AiDroid* outperforms others in real-time Android malware detection. *AiDroid* has already been incorporated into Tencent Mobile Security product that protects millions of users worldwide.

Acknowledgments

Y. Ye, S. Hou, L. Chen's work is partially supported by the NSF under grants CNS-1618629, CNS-1814825, CNS-1845138 and OAC-1839909, the DoJ/NIJ under grant NIJ 2018-75-CX-0032, the WV HEPC Grant (HEPC.dsr.18.5), and the WVU RSA grant (R-844).

References

[Chang *et al.*, 2015] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. Heterogeneous network embedding via deep architectures. In *KDD*, pages 119–128. ACM, 2015.

[Chen *et al.*, 2017] Lingwei Chen, Shifu Hou, and Yanfang Ye. Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks. In *ACSAC*, pages 362–372. ACM, 2017.

[Chen *et al.*, 2018] Hongxu Chen, Hongzhi Yin, Weiqing Wang, Hao Wang, Quoc Viet Hung Nguyen, and Xue Li. Pme: Projected metric embedding on heterogeneous networks for link prediction. In *KDD*, 2018.

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

[Fan *et al.*, 2018] Yujie Fan, Shifu Hou, Yiming Zhang, Yanfang Ye, and Melih Abdulhayoglu. Gotcha-sly malware!: Scorpion a metagraph2vec based malware detection system. In *KDD*, pages 253–262, 2018.

[Fu *et al.*, 2017] Tao-Yang Fu, Wang-Chien Lee, and Zhen Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *CIKM*, pages 1797–1806, 2017.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.

[Hou *et al.*, 2016] Shifu Hou, Aaron Saas, Yanfang Ye, and Lifei Chen. Droiddelver: An android malware detection system using deep belief network based on api call blocks. In *WAIM*, pages 54–66, 2016.

[Hou *et al.*, 2017] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *KDD*, 2017.

[Huang *et al.*, 2018] Sheng-Jun Huang, Jia-Wei Zhao, and Zhao-Yang Liu. Cost-effective training of deep cnns with active model adaptation. In *KDD*, 2018.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

[Ma *et al.*, 2018] Jianxin Ma, Peng Cui, and Wenwu Zhu. Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks. In *AAAI*, 2018.

[Mikolov *et al.*, 2013a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *arXiv preprint arXiv:1301.3781*, 2013.

[Mikolov *et al.*, 2013b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710, 2014.

[Saracino *et al.*, 2018] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, and Fabio Martinelli. Madam: Effective and efficient behavior-based android malware detection and prevention. *TDSC*, pages 240–253, 2018.

[Statcounter, 2018] Statcounter. Mobile operating system market share worldwide. In *http://gs.statcounter.com/os-market-share /mobile/worldwide*, 2018.

[Sun and Han, 2012] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 2012.

[Sutskever *et al.*, 2014] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[Szegedy *et al.*, 2015] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[Tang *et al.*, 2015] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, 2015.

[TencentSecurity, 2018] TencentSecurity. Mobile phone security report for the first half of 2018. In *https://m.qq.com /security_lab/news _detail_471.html*, 2018.

[Wu and Hung, 2014] Wen-Chieh Wu and Shih-Hao Hung. Droiddolphin: A dynamic android malware detection framework using big data and machine learning. In *RACS*, 2014.

[Wu *et al.*, 2012] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, and K. P. Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *Asia JCIS*, pages 62–69, Aug 2012.

[Ye *et al.*, 2010] Yanfang Ye, Tao Li, Qingshan Jiang, and Youyu Wang. Cimds: adapting postprocessing techniques of associative classification for malware detection. *TSMC*, 40(3):298–307, 2010.

[Ye *et al.*, 2011] Yanfang Ye, Tao Li, Shenghuo Zhu, Weiwei Zhuang, Egemen Tas, Umesh Gupta, and Melih Abdulhayoglu. Combining file content and file relations for cloud based malware detection. In *KDD*, 2011.

[Ye *et al.*, 2017] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. A survey on malware detection using data mining techniques. *CSUR*, 50(3):40, 2017.

[Ye *et al.*, 2018] Yanfang Ye, Shifu Hou, Lingwei Chen, Xin Li, Liang Zhao, Shouhuai Xu, Jiabin Wang, and Qi Xiong. Icsd: An automatic system for insecure code snippet detection in stack overflow over heterogeneous information network. In *ACSAC*, pages 542–552, 2018.

[Zhao *et al.*, 2018] Danling Zhao, Jichao Li, Yuejin Tan, Kewei Yang, Bingfeng Ge, and Yajie Dou. Optimization adjustment of human resources based on dynamic heterogeneous network. *Physica A*, 2018.

[Zhu and Ghahramani, 2002] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. *CMU Technical report*, 2002.