iDev: Enhancing Social Coding Security by Cross-platform User Identification Between GitHub and Stack Overflow

Yujie Fan[1,2], **Yiming Zhang**[1,2], **Shifu Hou**[1,2], **Lingwei Chen**[2], **Yanfang Ye**[1,2*]
Chuan Shi[3], **Liang Zhao**[4] and **Shouhuai Xu**[5]
[1]Department of CDS, Case Western Reserve University, OH, USA
[2]Department of CSEE, West Virginia University, WV, USA
[3]School of CS, Beijing University of Posts and Telecommunications, Beijing, China
[4]Department of IST, George Mason University, VA, USA
[5]Department of CS, University of Texas at San Antonio, TX, USA

Abstract

As modern social coding platforms such as GitHub and Stack Overflow become increasingly popular, their potential security risks increase as well (e.g., risky or malicious codes could be easily embedded and distributed). To enhance the social coding security, in this paper, we propose to automate cross-platform user identification between GitHub and Stack Overflow to combat the attackers who attempt to poison the modern software programming ecosystem. To solve this problem, an important insight brought by this work is to leverage social coding properties in addition to user attributes for cross-platform user identification. To depict users in GitHub and Stack Overflow (attached with attributed information), projects, questions and answers as well as the rich semantic relations among them, we first introduce an attributed heterogeneous information network (AHIN) for modeling. Then, we propose a novel AHIN representation learning model *AHIN2Vec* to efficiently learn node (i.e., user) representations in AHIN for cross-platform user identification. Comprehensive experiments on the data collections from GitHub and Stack Overflow are conducted to validate the effectiveness of our developed system *iDev* integrating our proposed method in cross-platform user identification by comparisons with other baselines.

1 Introduction

As computing devices and the Internet become increasingly ubiquitous, software has played a vital role in people's everyday lives. Recently, there has been an exponential growth in the number of software whose market has grown into a multi-billion dollars industry [Statista, 2018]. Unlike conventional software development ecosystem where developers significantly rely on code handbooks to create software from scratch, more and more software products are now created with the support from a highly interoperable and collaborative social coding platforms consisting of *social coding repositories* (e.g., GitHub - the largest source code repository hosting more than 100 million software projects maintained by over 31 million registered developers [GitHub, 2019]) and *online programming discussion platforms* (e.g., Stack Overflow - the largest online programming discussion platform with about 50 million visits each month [StackOverflow, 2018]). Within this software programming ecosystem, developers can reuse code snippets and libraries or adapt existing ready-to-use projects to expedite software development. However, the popularity and openness of such social coding environment not only attract developers to contribute legitimate codes but also attackers to disseminate malicious codes through ready-to-use applications [Ye *et al.*, 2018]. For example, according to a recent study [Chen *et al.*, 2016], 117 potentially harmful libraries on Android scattered in GitHub were found to infect 98,308 Google Play apps, which contain risky behaviors such as stealthily recording audio and video. To maintain a safe and productive ecosystem against malicious attacks, GitHub provides a security bug bounty site for code vulnerability reporting [Ducklin, 2017]. Nevertheless, such security policy is limited and there still lacks a principled approach toward addressing those security-related concerns. For example, from a social engineering perspective, humans are generally thought to be the weakest link in the security chain - *attackers could utilize the interplay between GitHub and Stack Overflow to poison the overall ecosystem of software development*.

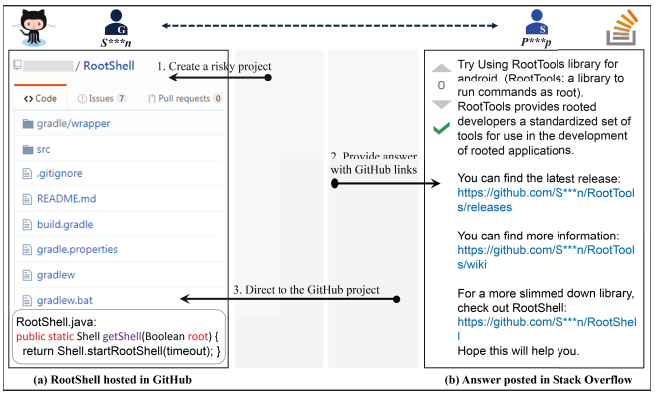

Figure 1: Interplay between GitHub and Stack Overflow.

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

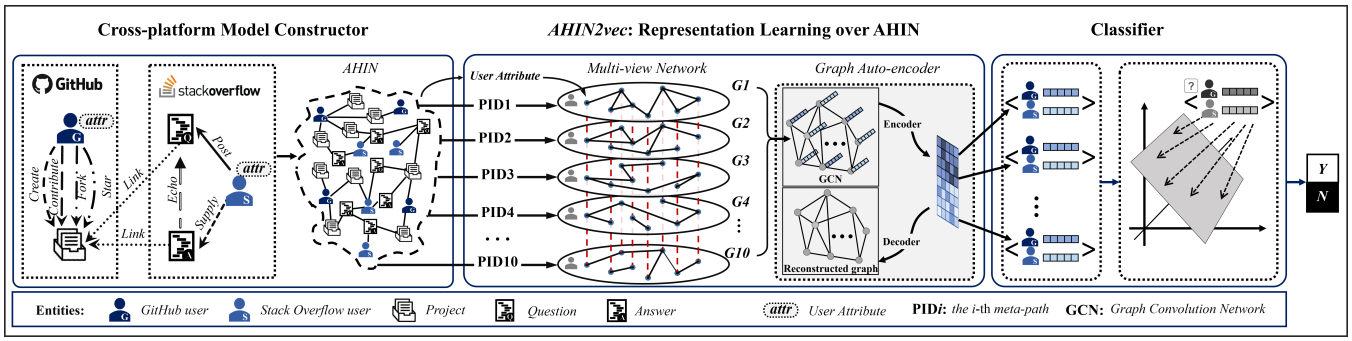

Figure 2: System overview of *iDev* for cross-platform user identification between GitHub and Stack Overflow.

Recent studies [Vasilescu *et al.*, 2013; Silvestri *et al.*, 2015; Baltes *et al.*, 2017] have shown that the interplay between the development process in GitHub and Stack Overflow activities is more active than we had thought. That is, GitHub committers often ask/answer questions on Stack Overflow; meanwhile, they can also engage in Stack Overflow to provide their knowledge and codes to others. This observation has the subtle implication from a security point of view: attackers might deliberately post questions/answers on Stack Overflow to lure innocent users to download and embed malicious codes hosted in GitHub. As shown in Figure 1: user "S***n" provides a risky *RootShell* library in GitHub which offers a root access to Android devices (Figure 1.(a)); to promote this code, the user with username "P***p" posts an answer thread in Stack Overflow, including the link directing into the *RootShell* library hosted in GitHub, to answer a question related to system file copy (Figure 1.(b)). In such case, the questioner or other less experienced users may be directed into using the alleged solution without maintaining a healthy skepticism; once they reuse this library to generate the production software, the victim devices might be compromised (e.g., system operations being disrupted, or sensitive information being leaked). This finding implies the importance of cross-platform user identification to detect attackers who attempt to poison the modern software programming ecosystem. Given the large number of user accounts in the growing GitHub and Stack Overflow, it is simply impossible to manually link suspicious accounts for cross-platform user identification. Though some users in Stack Overflow do provide their GitHub links, this portion is very small (i.e., about 0.44% based on our data collections described in Section 3.1). To automate the process, unfortunately, there have been few works with the exception of [Silvestri *et al.*, 2015] which merely used user attributes (e.g., user profiles) for identification. Therefore, to combat the attacks in modern software programming ecosystem and to enhance its security, it is highly desirable to develop novel methodologies that can automate cross-platform user identification between GitHub and Stack Overflow.

To address the above challenges, an important insight brought by this work is to leverage social coding properties in addition to user attributes for cross-platform user identification. As social coding platforms, both GitHub and Stack Overflow are characterized by user activities and communications, i.e., a rich source of heterogeneous information are available including users, projects, questions, answers, and their semantic relations. To utilize such social coding properties and user attributes, in this paper, we first introduce an attributed heterogeneous information network (AHIN) [Li *et al.*, 2017] for modeling. Then, we propose *AHIN2Vec* to efficiently learn node (i.e., user) representations in AHIN for cross-platform user identification. We develop a system called *iDev* (shown in Figure 2) which integrates our proposed method for cross-platform user identification between GitHub and Stack Overflow. *iDev* has the following merits:

- It introduces AHIN as an abstract representation of GitHub and Stack Overflow data for the first time.
- It presents a novel yet elegant AHIN embedding model (i.e., *AHIN2Vec*) to efficiently learn low-dimensional representations for nodes (i.e., users) in AHIN.
- Comprehensive experimental studies demonstrate the performance of *iDev* in cross-platform user identification for enhancing social coding security.

2 Proposed Method

In this section, we introduce the proposed method integrated in *iDev* for cross-platform user identification in detail.

2.1 Feature Extraction

User Attributed Features. To fingerprint a user, we consider both his/her profile information and posted text content. In GitHub and Stack Overflow, user profile plays an important role in resolving his/her identity. Thus, we extract three kinds of features from user profile to depict each user: *username*, *location* and *contact information* (e.g., email address, twitter account). Note that, for username, we first apply standard string matching techniques to measure the similarity of two usernames; if their similarity is greater than a user-specific threshold, we regard these two usernames as the same (e.g., "D**Tomas" and "D**T0mas"). Then, we apply one-hot encoding to convert the extracted features to a binary feature vector. To obtain better user representations, for GitHub user, we further consider the description of each project he/she creates; for Stack Overflow user, we consider his/her posted questions and answers. To deal with such text content, we propose to exploit *doc2vec* [Le and Mikolov, 2014] to convert each text of variant size into a fixed-length feature vector (i.e., we empirically set it as 100). Accordingly, we concatenate these two feature vectors to form an attributed feature vector for each user.

Social Coding Properties. To comprehensively depict users in GitHub and Stack Overflow, we not only utilize the above attributed features, but consider the rich social coding properties among them including: (1) ***GitHub relations***: i) *R1*: the *user-create-project* relation describes whether a GitHub user creates a project; ii) *R2*: *user-star-project* relation means a user stars a project, denoting his/her interest to keep track of the project; iii) *R3*: *user-fork-project* relation denotes a user either proposes changes to someone else's project or uses someone else's project as a starting point for his/her own idea; iv) *R4*: the *user-contribute-project* relation describes if a user contributes to a project. (2) ***Stack Overflow relations***: the *user-post-question* relation (*R5*) and the *user-supply-answer* relation (*R6*) denote if a user posts a question or supplies an answer respectively; to denote the Q&A relationship, we build the *answer-echo-question* relation (*R7*) to indicate if an answer responds (i.e., echoes) to a particular question. (3) ***Cross-platform relations***: a question or answer thread may have a link directing into a GitHub project, we accordingly extract the *question-link-project* relation (*R8*) and the *answer-link-project* relation (*R9*) for representations.

2.2 AHIN Construction

Though heterogeneous information network (HIN) [Sun *et al.*, 2011] has shown the success of modeling different types of entities and relations, it has limited capability of modeling additional attributes attached to entities. To solve this problem, we present attributed HIN (AHIN) for modeling.

Definition 1 ***Attributed Heterogeneous Information Network (AHIN)*** *[Li* et al.*, 2017]. Let $\mathcal{T} = \{T_1, ..., T_m\}$ be a set of m entity types. For each entity type T_i, let $\mathcal{X}_i$ be the set of entities of type T_i and A_i be the set of attributes defined for entities of type T_i. An entity x_j of type T_i is associated with an attribute vector $\mathbf{f}_j = (f_{j1}, f_{j2}, ..., f_{j|A_i|})$. An AHIN is defined by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$ with an* entity *type mapping ϕ: $\mathcal{V} \to \mathcal{T}$ and a* relation *type mapping ψ: $\mathcal{E} \to \mathcal{R}$, where $\mathcal{V} = \bigcup_{i=1}^{m} \mathcal{X}_i$ denotes the entity set and $\mathcal{E}$ is the relation set, $\mathcal{T}$ denotes the entity type set and $\mathcal{R}$ is the relation type set, $\mathcal{A} = \bigcup_{i=1}^{m} A_i$, and the number of entity types $|\mathcal{T}| > 1$ or the number of relation types $|\mathcal{R}| > 1$. The **network schema** for an AHIN $\mathcal{G}$, denoted by $\mathcal{T}_\mathcal{G} = (\mathcal{T}, \mathcal{R})$, is a graph with nodes as entity types from $\mathcal{T}$ and edges as relation types from $\mathcal{R}$.*

For our case, we have five types of entities and nine types of relations among them while nodes with entity type of user are further attached with an extracted feature vector representing its associated attributes. Based on the definition above, the network schema for AHIN in our application is shown in Figure 3.(a). To formulate the higher-order relationships among entities, the concept of meta-path [Sun *et al.*, 2011] has been proposed: a ***meta-path*** $\mathcal{P}$ is a path defined on the 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 the length of $\mathcal{P}$. In our application, we design ten meaningful meta-paths (i.e., *PID1-PID10* shown in Figure 3.(b)) to jointly characterize the relatedness between two users in GitHub and Stack Overflow from different views.

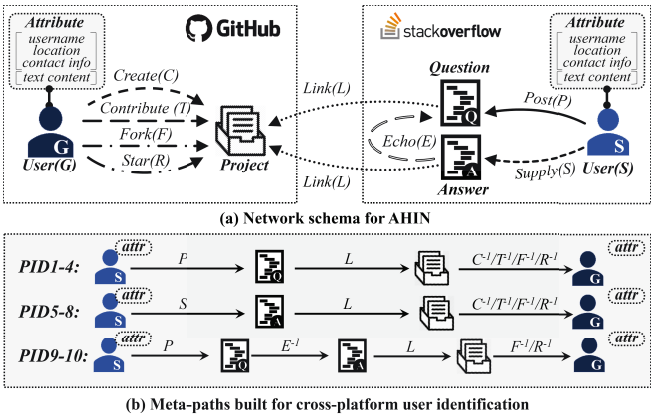

Figure 3: Network schema and meta-paths for AHIN.

PID5 is one of the examples: $user(S) \xrightarrow{supply} answer \xrightarrow{link} project \xrightarrow{create^{-1}} user(G)$ which denotes that a user in Stack Overflow is connected with a user in GitHub if the Stack Overflow $user(S)$ supplies an answer including a link directing into a project created by the Github $user(G)$.

2.3 AHIN2Vec

Given an AHIN $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$, the ***AHIN representation learning*** [Dong *et al.*, 2017] task is to learn a function $f : \mathcal{V} \to \mathbb{R}^D$ 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 of preserving both structural and semantic relations among them. Although many network embedding methods [Perozzi *et al.*, 2014; Tang *et al.*, 2015; Dong *et al.*, 2017; Fu *et al.*, 2017] have been proposed recently, few of them deal with node embeddings in AHIN. Here, we propose a novel AHIN embedding model *AHIN2Vec* to learn node representations: we first map the constructed AHIN to a multi-view network consisting of a set of single-view attributed graphs; we then efficiently fuse such single-view graphs into an unified attributed graph via subspace analysis on Grassmann manifolds; afterwards, we propose a graph auto-encoder model to learn user embeddings for cross-platform user identification.

Multi-view network built from AHIN. We first define a ***multi-view network*** [Shi *et al.*, 2018] as $\mathcal{G}^\mathcal{M} = (\mathcal{V}, \{\mathcal{E}^i\}_{i=1}^M, \mathcal{A})$ consisting of a set $\mathcal{V}$ of nodes and M views, where $\mathcal{E}^i$ consists of all edges in view $i \in \{1, ..., M\}$. If a multi-view network is weighted, then there exists a weight mapping $w : \{\mathcal{E}^i\}_{i=1}^M \to \mathbb{R}$ such that $w_{vv'}^i := w(e_{vv'}^i)$ is the weight of the edge $e_{vv'}^i \in \mathcal{E}^i$, which joints nodes $v \in \mathcal{V}$ and $v' \in \mathcal{V}$ in view $i \in \{1, ..., M\}$. Based on this definition, we then map the constructed AHIN to a multi-view network consisting of a set of single-view attributed graphs encoding the relatedness over users depicted by different meta-paths. More specifically, given an AHIN $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$ and M meta-paths, a multi-view network with M single-view attributed graphs $\mathcal{G}^i = (\mathcal{V}, \mathcal{E}^i, \mathcal{A})$ is built where the i-th view graph is generated based on meta-path $\mathcal{P}_i$ ($i = \{1, ..., M\}$). These single-view attributed graphs depict different kinds of interactions among users, which can reflect different-views of user representations. In our case, each node in $\mathcal{G}^i$ denotes a user

in GitHub or Stack Overflow and an edge between two users denotes if they can be connected under meta-path *PIDi*. The user feature matrix $\mathbf{X}$ in $\mathcal{G}^i$ can be represented as:

$$\mathbf{X} = \mathbf{u}_1 \oplus \mathbf{u}_2 \oplus \ ... \ \oplus \mathbf{u}_N, \tag{1}$$

where N is the number of users in GitHub and Stack Overflow, $\oplus$ is the concatenation operator, $\mathbf{u}_i \in \mathbb{R}^f$ is the f-dimensional attributed feature vector for the i-th user.

Fusing multiple single-view attributed graphs. Given a multi-view network with M single-view attributed graph $\mathcal{G}^i$ ($i = \{1, ..., M\}$), we aim to effectively fuse them into an unified graph via a two-phase procedure: representing each single-view graph in an individual subspace and then combining the multiple subspaces to generate an unified space.

Let $\mathbf{A}_i$ and $\mathbf{D}_i$ be the adjacency matrix and degree matrix of i-th view graph respectively, inspired by the spectral clustering [Ng *et al.*, 2002], we map i-th view graph into a subspace via solving the following trace minimization problem:

$$\min_{\mathbf{U}_i \in \mathbb{R}^{N \times k}} tr(\mathbf{U}_i^T \mathbf{L}_i \mathbf{U}_i), \text{s.t. } \mathbf{U}_i^T \mathbf{U}_i = \mathbf{I}. \tag{2}$$

where $\mathbf{L}_i = \mathbf{D}_i^{-\frac{1}{2}}(\mathbf{D}_i - \mathbf{A}_i)\mathbf{D}_i^{-\frac{1}{2}}$ is the normalized graph Laplacian; $\mathbf{U}_i$ contains the first k eigenvectors (corresponding to the k smallest eigenvalues) of $\mathbf{L}_i$. This optimization problem can be solved by the Rayleigh-Ritz theorem [Von Luxburg, 2007]. By adopting the above spectral embedding, we can define a meaningful subspace representation for each single-view graph. Then, to effectively make a combination of the learned multiple subspaces, following [Dong *et al.*, 2014], we propose to find a representative subspace that is close to all the individual subspaces and its representation $\mathbf{U}$ preserves the information about node connectivity in each single-view graph. Based on Grassmann manifold theory, the projection distance between two subspaces can be defined as a set of principal angles $\{\theta_i\}_{i=1}^k$ between these subspaces:

$$d^2_{proj}(\mathbf{U}_1, \mathbf{U}_2) = \sum_{i=1}^{k} \sin^2 \theta_i = k - tr(\mathbf{U}_1 \mathbf{U}_1^T \mathbf{U}_2 \mathbf{U}_2^T). \tag{3}$$

Accordingly, the projection distance between the target representative subspace $\mathbf{U}$ and the M individual subspaces $\{\mathbf{U}_i\}_{i=1}^M$ is calculated as:

$$d^2_{proj}(\mathbf{U}, \{\mathbf{U}_i\}_{i=1}^M) = kM - \sum_{i=1}^{M} tr(\mathbf{U}\mathbf{U}^T \mathbf{U}_i \mathbf{U}_i^T). \tag{4}$$

Minimization of Eq. (4) enforces the representative subspace to be close to all the individual subspaces. Further, to preserve the node connectivity in each single-view graph, we minimize the Laplacian quadratic form in Eq. (2). Finally, we solve the following optimization problem to fuse multiple subspaces:

$$\min_{\mathbf{U} \in \mathbb{R}^{N \times k}} \sum_{i=1}^{M} tr(\mathbf{U}^T \mathbf{L}_i \mathbf{U}) + \alpha[kM - \sum_{i=1}^{M} tr(\mathbf{U}\mathbf{U}^T \mathbf{U}_i \mathbf{U}_i^T)], \tag{5}$$

subject to $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, where α is a regularization parameter that balances the trade-off between the two terms. By ignoring constant terms and rearranging the trace form, we have:

$$\min_{\mathbf{U} \in \mathbb{R}^{N \times k}} tr[\mathbf{U}^T (\sum_{i=1}^{M} \mathbf{L}_i - \alpha \sum_{i=1}^{M} \mathbf{U}_i \mathbf{U}_i^T)\mathbf{U}]. \tag{6}$$

As stated before, this trace minimization problem can be solved by Rayleigh-Ritz theorem with a modified Laplacian:

$$\mathbf{L}_{mod} = \sum_{i=1}^{M} \mathbf{L}_i - \alpha \sum_{i=1}^{M} \mathbf{U}_i \mathbf{U}_i^T. \tag{7}$$

Graph auto-encoder. In order to learn robust node embeddings for the fused attributed graph, we propose to exploit the graph auto-encoder model [Kipf and Welling, 2016b] which consists of an encoder aiming at encoding graph data to a compact representation, and a decoder to reconstruct the topological graph information from such representation.

Encoder: Since graph convolutional network (GCN) has shown its power to capture the graph topological structure and the node attributed features [Kipf and Welling, 2016a], we employ GCN as an encoder to learn a latent representation. Let $\mathbf{A} \in \mathbb{R}^{N \times N}$ be the adjacency matrix generated from $\mathbf{L}_{mod}$ and $\mathbf{X}$ is the user feature matrix. Following the idea of GCN, the convolutional layer:

$$\mathbf{Z}^{l+1} = \sigma\left(\tilde{\mathbf{A}}\, \mathbf{Z}^l\, \mathbf{W}^l\right), \tag{8}$$

where $\tilde{\mathbf{A}}$ is a symmetric normalization of $\mathbf{A}$ with self-loop, i.e. $\tilde{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}} \hat{\mathbf{A}} \hat{\mathbf{D}}^{-\frac{1}{2}}$ with $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. Here $\mathbf{I}$ is the identity matrix and $\hat{\mathbf{D}}$ is the diagonal node degree matrix of $\hat{\mathbf{A}}$. The $\mathbf{Z}^l$ and $\mathbf{W}^l$ denote the l-th hidden layer and the layer-specific parameters, $\mathbf{Z}^0 = \mathbf{X}$ and σ denotes an activation function, such as the ReLU($\cdot$) = $\max(0, \cdot)$. In this paper, we employ a two-layer GCN as encoder:

$$\mathbf{Z} = q(\mathbf{Z}|\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\, \text{ReLU}\left(\tilde{\mathbf{A}}\, \mathbf{X}\, \mathbf{W}^0\right) \mathbf{W}^1. \tag{9}$$

Decoder: The decoder model is used to reconstruct graph topological structure $\mathbf{A}$. More specifically, we train a decoder to predict whether there is a link between two nodes based on the graph embedding learned from encoder:

$$p(\mathbf{A}|\mathbf{Z}) = \prod_{i=1}^{N} \prod_{i=1}^{N} p(\mathbf{A}_{ij}|\mathbf{z}_i, \mathbf{z}_j), \tag{10}$$

$$p(\mathbf{A}_{ij} = 1|\mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^T \mathbf{z}_j), \tag{11}$$

where $\sigma(x) = 1/(1 + e^x)$ is a sigmoid function.

Optimization: Based on the description above, we minimize the reconstruction error as following:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}|\mathbf{Z})]. \tag{12}$$

We then perform stochastic gradient descent for training.

Algorithm 1 illustrates our proposed AHIN representation learning model *AHIN2Vec*. After employing *AHIN2Vec*, we apply average pooling on each pair of GitHub and Stack Overflow user embeddings, which are then fed to the classifier (i.e., we employ support vector machine (SVM) in our application) for cross-platform user identification.

3 Experimental Results and Analysis

In this section, we fully evaluate the performance of our developed system *iDev* by comparisons with other baselines.

Algorithm 1: *AHIN2Vec*.

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$: AHIN, $\{\mathcal{P}_i\}_{i=1}^{M}$: meta-paths, α: regularization parameter; T: the number of iterations.
Output: $\mathbf{Z}$: node representations for AHIN.

for $i = 1 \rightarrow M$ **do**
  Generate single-view graph $\mathcal{G}^i$ based on $\mathcal{P}_i$;
  Learn the subspace representation $\mathbf{U}_i$ for $\mathcal{G}^i$ via solving Eq. (2);
end
Compute the merged Laplacian $\mathbf{L}_{mod}$ using Eq. (7);
for $i = 1 \rightarrow T$ **do**
  Generate latent representations $\mathbf{Z}$ through Eq. (9);
  Update the graph auto-encoder with its stochastic gradient by Eq.(12);
end
Return $\mathbf{Z}$;

3.1 Data Collection

We collect the data from GitHub and Stack Overflow. For Stack Overflow, we collect its public data dump [StackExchange, 2019] including 9,737,249 users; for those Stack Overflow users who provide the GitHub links in their profiles (i.e., 42,840 users), we develop a set of crawling tools to further obtain the related information in GitHub. We obtain the ground-truth under a pseudo setting: we first randomly subsample 10% Stack Overflow users who provides GitHub links (i.e., 4,284 users) and regard these 4,284 user pairs as positive examples, then randomly match each Stack Overflow user to a GitHub user to generate 4,284 negative examples. In this pseudo setting, the contact information attached to each user is set as null. Based on the extracted features and designed network schema, the constructed AHIN has 25,875 nodes (i.e., 4,284 nodes with type of GitHub user, 4,864 nodes with type of project, 4,284 nodes with type of Stack Overflow user, 2,184 nodes with type of question, and 10,259 with type of answer) and 75,824 edges including relations of *R1-R9*. In the following experiments, we use accuracy (i.e., ACC) and F1 measure for evaluations.

3.2 Baseline Methods

We validate the performance of our proposed method for cross-platform user identification in GitHub and Stack Overflow by comparisons with different groups of baseline methods. First, we evaluate different types of features:

- **User Attributes.** This category only considers the extracted user attributed features described in Section 2.1, denoted as ***f-1***;
- **Different AHIN-based Features.** This type of features augment user attributes with the AHIN-based relations in three different ways: (1) ***f-2*** concatenates user attributes and relations represented by the best performed single-view graph; (2) ***f-3*** concatenates user attributes and relations represented by simply merging the edges of ten different single-view graphs; and (3) ***f-4*** concatenates user attributes and relations which are formulated by employing our proposed fusion method to merge different single-view graphs into an unified one.

Second, we evaluate our proposed AHIN embedding method *AHIN2Vec* by comparisons with following baselines.

- **DeepWalk** [Perozzi *et al.*, 2014] learns node vectors by capturing node pairs within w-hop neighborhood via uniform random walks in the network.
- **LINE** [Tang *et al.*, 2015] learns embeddings by preserving first and second order proximities between nodes.
- **metapath2vec** [Dong *et al.*, 2017] embeds the semantic information based on a single meta-path.
- **HIN2Vec** [Fu *et al.*, 2017] learns embeddings to capture rich relation semantics in HIN via a neural network.

For DeepWalk and LINE, we ignore the heterogeneous property and directly feed the network for embedding. Note that, since all baselines are incapable of dealing with attributed features attached to the nodes (i.e., users), here we simply concatenate the node embedding learned by each method with the user attributed feature vector to represent a user. In our method, we train graph auto-encoder as suggested in [Kipf and Welling, 2016b]. To facilitate the comparisons, we use the experimental procedure as in [Perozzi *et al.*, 2014; Tang *et al.*, 2015; Dong *et al.*, 2017]: we set vector dimension $d = 200$ and randomly select a portion of data (ranging from 10% to 90%) for training and the remaining one for testing.

3.3 Comparisons and Analysis

Based on our data collection, we show the comparison results for all the baseline methods introduced above for cross-platform user identification. We first evaluate the performance of different types of features using ten-fold cross validations. From Table 1, we can observe that different features show different performances: (1) feature engineering (***f-2***, ***f-3*** and ***f-4***) helps the performance of machine learning since the rich semantics encoded in AHIN-based relations can bring more information; (2) two augmented features formulated from the merged graphs (***f-3*** and ***f-4***) outperform ***f-2*** using user relations extracted from a single-view graph; further, the proposed method to fuse different single-views (***f-4***) indeed performs better than the simple graph merging (***f-3***); (3) our proposed method achieves the best result. The reason behind this is that *iDev* leverages user attributes and AHIN-based relations in an expressive and comprehensive way, which is more informative than the simple augmented features.

Metric	Attrs	Attrs+Relations			*iDev*
	f-1	***f-2***	***f-3***	***f-4***	
ACC	0.868	0.902	0.916	0.921	0.936
F1	0.863	0.898	0.915	0.917	0.934

Table 1: Comparisons of different types of features

We also show the comparisons in Table 2 for different network embedding models. From the results, we can see that our proposed *AHIN2Vec* outperforms all baselines in terms of *ACC* and *F1*. That is to say, *AHIN2Vec* learns significantly

better node (i.e., user) representations in AHIN than current state-of-the-art methods. The success of *AHIN2Vec* lies in: (1) the proper consideration and accommodation of the heterogeneous property of AHIN (i.e., the multiple types of nodes and relations), (2) the advantage of graph auto-encoder to incorporate both graph structure and node attributes for representation learning.

Metric	Method	10%	30%	50%	70%	90%
ACC	Deepwalk	0.812	0.854	0.873	0.892	0.906
	Line	0.823	0.864	0.874	0.892	0.909
	metapath2vec	0.849	0.870	0.891	0.913	0.929
	Hin2vec	0.855	0.875	0.897	0.915	0.926
	AHIN2Vec	0.866	0.886	0.911	0.919	0.939
F1	Deepwalk	0.809	0.850	0.869	0.889	0.904
	Line	0.821	0.860	0.871	0.889	0.906
	metapath2vec	0.846	0.867	0.888	0.911	0.927
	Hin2vec	0.851	0.871	0.894	0.912	0.924
	AHIN2Vec	0.862	0.882	0.908	0.916	0.938

Table 2: Comparisons of different embedding methods

3.4 Parameter Sensitivity and Scalability

In this set, we first conduct parameter sensitivity analysis of how different choices of dimension d will affect the performance of *iDev* in cross-platform user identification by ten-fold cross validations. As shown in Figure 4.(a), *iDev* is not strictly sensitive to dimension d and is able to reach high performance under a cost-effective parameter choice. For scalability, we further evaluate the running time of *iDev* with different sizes of the dataset. Figure 4.(b) shows that the running time is quadratic to the number of sampled users. When dealing with more data, parallel algorithms can be developed.

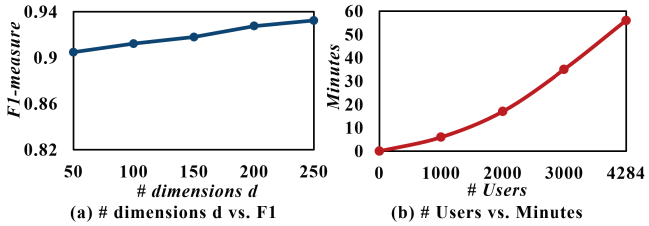

Figure 4: Parameter sensitivity and scalability

3.5 Case Studies

To better understand and gain deeper insights into the security-related risks of modern social coding ecosystem, we further apply our developed system *iDev* for cross-platform user identification in the wild. For the identified user pairs, we sample 565 pairs and validate them using conclusive evidences. Among these 565 pairs, 502 pairs (88.85%) are with high confidence that they are the same individuals and 15 pairs are uncertain (2.65%). For example, one of our identified user pairs "a***2" in GitHub and "a***r" in Stack Overflow have different usernames, locations and contact information; meanwhile the user profile in Stack Overflow doesn't provide GitHub link. However, "a***r" in Stack Overflow posts a question linking into a GitHub project which offers OTA updater for unofficial ROMs created by "a***2". After further investigation, we find that these two users are the same one in developing this GitHub project. The studies based on the identified user pairs could help understand and thus prevent the dissemination of risky or malicious codes cross different social coding platforms.

4 Related Work

There have been many works on enhancing code security such as malicious code detection [Ye *et al.*, 2007; Ye *et al.*, 2011; Hou *et al.*, 2017; Ye *et al.*, 2017; Chen *et al.*, 2017; Fan *et al.*, 2018; Ye *et al.*, 2018]. With the popularity of social coding platforms, there have been several studies on the interplay between GitHub and Stack Overflow [Vasilescu *et al.*, 2013; Silvestri *et al.*, 2015; Baltes *et al.*, 2017]. Though the results are promising, most of them fail to consider the social coding properties in solving their problems. Different from the existing works, in this paper, to conduct cross-platform user identification, we propose to utilize not only user attributes, but also various kinds of relationships among users, projects, questions and answers.

HIN has shown the success of modeling different types of entities and relations [Sun *et al.*, 2011], but it has limited capability of modeling additional attributes attached to entities. To tackle this challenge, AHIN is proposed for representation [Li *et al.*, 2017]. To better address representation learning for HIN, many efficient network embedding methods have been proposed such as metapath2vec [Dong *et al.*, 2017], HIN2vec [Fu *et al.*, 2017]. However, these models are unable to deal with the attributed information associated with the entity. To address this issue, we propose *AHIN2Vec* to learn the desirable node representations in AHIN.

5 Conclusion

We develop a system named *iDev* to automate cross-platform user identification between GitHub and Stack Overflow. In *iDev*, to depict a rich source of heterogeneous information and user attributes, we present a structural AHIN for representation and use a meta-path based approach to characterize the semantic relatedness over users. To efficiently learn node representations in AHIN, we propose a novel AHIN embedding model *AHIN2Vec* which first maps the constructed AHIN to a multi-view network, then applies subspace analysis to obtain an unified attributed graph and later exploits graph auto-encoder to learn the node embeddings. After that, we apply average pooling on each pair of GitHub and Stack Overflow user embeddings which are then fed to the classifier for cross-platform user identification. The promising experimental results on the collected datasets demonstrate that *iDev* outperforms other baseline methods.

Acknowledgments

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

References

[Baltes *et al.*, 2017] Sebastian Baltes, Richard Kiefer, and Stephan Diehl. Attribution required: Stack overflow code snippets in github projects. In *ICSE*, pages 161–163, 2017.

[Chen *et al.*, 2016] Kai Chen, Xueqiang Wang, Yi Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Bin Ma, Aohui Wang, Yingjun Zhang, and Wei Zou. Following devil's footprints: Cross-platform analysis of potentially harmful libraries on android and ios. In *S&P*, pages 357–376. IEEE, 2016.

[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.

[Dong *et al.*, 2014] Xiaowen Dong, Pascal Frossard, Pierre Vandergheynst, and Nikolai Nefedov. Clustering on multi-layer graphs via subspace analysis on grassmann manifolds. *IEEE Trans. Signal Process*, 62(4):905–918, 2014.

[Dong *et al.*, 2017] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*, pages 135–144, 2017.

[Ducklin, 2017] Paul Ducklin. Github starts scanning millions of projects for insecure components. https://nakedsecurity.sophos.com/2017/11/21/github-starts-scanning-millions-of-projects-for-insecure-components/, 2017. Accessed February 11, 2019.

[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. ACM, 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. ACM, 2017.

[GitHub, 2019] GitHub. Github: Built for developers. https://github.com/about, 2019. Accessed February 11, 2019.

[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*, pages 1507–1515. ACM, 2017.

[Kipf and Welling, 2016a] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*, 2016.

[Kipf and Welling, 2016b] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[Le and Mikolov, 2014] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, pages 1188–1196, 2014.

[Li *et al.*, 2017] Xiang Li, Yao Wu, Martin Ester, Ben Kao, Xin Wang, and Yudian Zheng. Semi-supervised clustering in attributed heterogeneous information networks. In *WWW*, pages 1621–1629, 2017.

[Ng *et al.*, 2002] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856, 2002.

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

[Shi *et al.*, 2018] Yu Shi, Fangqiu Han, Xinwei He, Xinran He, Carl Yang, Jie Luo, and Jiawei Han. mvn2vec: Preservation and collaboration in multi-view network embedding. *arXiv:1801.06597*, 2018.

[Silvestri *et al.*, 2015] Giuseppe Silvestri, Jie Yang, Alessandro Bozzon, and Andrea Tagarelli. Linking accounts across social networks: the case of stackoverflow, github and twitter. In *KDWeb*, pages 41–52, 2015.

[StackExchange, 2019] StackExchange. Stackexchange data dump. https://archive.org/details/stackexchange, 2019. Accessed January 5, 2019.

[StackOverflow, 2018] StackOverflow. Developer survey results 2018. https://insights.stackoverflow.com/survey/2018/, 2018. Accessed February 11, 2019.

[Statista, 2018] Statista. Statistics and market data on software. https://www.statista.com/markets/418/topic/484/software/, 2018. Accessed February 11, 2019.

[Sun *et al.*, 2011] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB Endowment*, 4(11):992–1003, 2011.

[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*, pages 1067–1077, 2015.

[Vasilescu *et al.*, 2013] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *ICSC*, pages 188–195. IEEE, 2013.

[Von Luxburg, 2007] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[Ye *et al.*, 2007] Yanfang Ye, Dingding Wang, Tao Li, and Dongyi Ye. Imds: Intelligent malware detection system. In *KDD*, pages 1043–1047. ACM, 2007.

[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*, pages 222–230. ACM, 2011.

[Ye *et al.*, 2017] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. A survey on malware detection using data mining techniques. *ACM CSUR*, 50(3):41, 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. ACM, 2018.